MySQL 3.23, 4.0, 4.1 Reference Manual

MySQL 3.23, 4.0, 4.1 Reference Manual

Abstract

This is the MySQL™ Reference Manual. It documents MySQL 3.23 through MySQL 4.1.25.

MySQL 3.23, 4.0, and 4.1 features. This manual describes features that are not included in every edition of MySQL 3.23, MySQL 4.0, and MySQL 4.1; such features may not be included in the edition of MySQL 3.23, MySQL 4.0, or MySQL 4.1; licensed to you. If you have any questions about the features included in your edition of MySQL 3.23, MySQL 4.0, or MySQL 4.1, refer to your MySQL 3.23, MySQL 4.0, or MySQL 4.1 agreement or contact your Oracle sales representative.

For release notes detailing the changes in each release of MySQL 3.23, 4.0, and 4.1, see Appendix C, *MySQL Release Notes*.

Document generated on: 2014-02-17 (revision: 37744)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Table of Contents

Preface and Legal Notice

This is the Reference Manual for the MySQL Database System, version 4.1, through release 4.1.25. It is also applicable for versions of the MySQL software previous to 4.1 (such as 3.23 or 4.0) because functional changes are indicated with reference to version numbers. For later MySQL releases, see the appropriately numbered edition of this manual. For example, if you are using a version 5.0 release of the MySQL software, please refer to the *MySQL 5.0 Reference Manual*.

If you are using MySQL 5.0, please refer to the *MySQL 5.0 Reference Manual*. If you are using MySQL 5.1, please refer to the *MySQL 5.1 Reference Manual*.

Legal Notices

Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle or as specifically provided below. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

For more information on the terms of this license, or for details on how the MySQL documentation is built and produced, please visit MySQL Contact & Questions.

For additional licensing information, including licenses for third-party libraries used by MySQL products, see Preface and Legal Notice.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Chapter 1 General Information

Table of Contents

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

The following list describes some sections of particular interest in this manual:

- For a discussion about the capabilities of the MySQL Database Server, see Section 1.3.2, “The Main Features of MySQL”.
- For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*. For information about upgrading MySQL, see Section 2.11.1, “Upgrading MySQL”, and the change notes at Appendix C, *MySQL Release Notes*.
- For a tutorial introduction to the MySQL Database Server, see Chapter 3, *Tutorial*.
- For information about configuring and administering MySQL Server, see Chapter 5, *MySQL Server Administration*.
- For information about setting up replication servers, see Chapter 14, *Replication*.
- For a list of currently known bugs and misfeatures, see Section B.5.8, “Known Issues in MySQL”.
- For a list of all the contributors to this project, see Section 1.10, “Credits”.
- For a history of new features and bugfixes, see Appendix C, *MySQL Release Notes*.
- For benchmarking information, see the `sql-bench` benchmarking directory in your MySQL distribution.

Important

To report problems or bugs, please use the instructions at Section 1.8, “How to Report Bugs or Problems”. If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support.

1.1 About This Manual

This is the Reference Manual for the MySQL Database System, version 4.1, through release 4.1.25. It is also applicable for versions of the MySQL software previous to 4.1 (such as 3.23 or 4.0) because functional changes are indicated with reference to version numbers. For later MySQL releases, see the appropriately numbered edition of this manual. For example, if you are using a version 5.0 release of the MySQL software, please refer to the *MySQL 5.0 Reference Manual*.

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at http://dev.mysql.com/doc/. Other formats also are available there, including HTML, PDF, and Windows CHM versions.

The Reference Manual source files are written in DocBook XML format. The HTML version and other formats are produced automatically, primarily using the DocBook XSL stylesheets. For information about DocBook, see http://docbook.org/

If you have questions about using MySQL, you can ask them using our mailing lists or forums. See Section 1.7.1, “MySQL Mailing Lists”, and Section 1.7.2, “MySQL Community Support at the MySQL Forums”. If you have suggestions concerning additions or corrections to the manual itself, please send them to the http://www.mysql.com/company/contact/.

This manual was originally written by David Axmark and Michael “Monty” Widenius. It is maintained by the MySQL Documentation Team, consisting of Paul DuBois, Stefan Hinz, Philip Olson, Daniel Price, Daniel So, and Jon Stephens.

1.2 Typographical and Syntax Conventions

This manual uses certain typographical conventions:

- `Text in this style` is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: “To reload the grant tables, use the `FLUSH PRIVILEGES` statement.”
- **`Text in this style`** indicates input that you type in examples.
- `Text in this style` indicates the names of executable programs and scripts, examples being `mysql` (the MySQL command-line client program) and `mysqld` (the MySQL server executable).
- *`Text in this style`* is used for variable input for which you should substitute a value of your own choosing.
- *Text in this style* is used for emphasis.
- **Text in this style** is used in table headings and to convey especially strong emphasis.
- `Text in this style` is used to indicate a program option that affects how the program is executed, or that supplies information that is needed for the program to function in a certain way. *Example*: “The `--host` option (short form `-h`) tells the `mysql` client program the hostname or IP address of the MySQL server that it should connect to”.
- File names and directory names are written like this: “The global `my.cnf` file is located in the `/etc` directory.”
- Character sequences are written like this: “To specify a wildcard, use the ‘`%`’ character.”

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, `shell>` indicates a command that you execute from your login shell, `root-shell>` is similar but should be executed as `root`, and `mysql>` indicates a statement that you execute from the `mysql` client program:

```
shell> type a shell command here
root-shell> type a shell command as root here
mysql> type a mysql statement here
```

In some areas different systems may be distinguished from each other to show that commands should be executed in two different environments. For example, while working with replication the commands might be prefixed with `master` and `slave`:

```
master> type a mysql command on the replication master here
slave> type a mysql command on the replication slave here
```

The “shell” is your command interpreter. On Unix, this is typically a program such as `sh`, `csh`, or `bash`. On Windows, the equivalent program is `command.com` or `cmd.exe`, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses *`db_name`*, *`tbl_name`*, and *`col_name`*. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (“`[`” and “`]`”) indicate optional words or clauses. For example, in the following statement, `IF EXISTS` is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (“`|`”). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (“`[`” and “`]`”):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (“`{`” and “`}`”):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (`...`) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, `SELECT ... INTO OUTFILE` is shorthand for the form of `SELECT` statement that has an `INTO OUTFILE` clause following other parts of the statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple *`reset_option`* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the `CC` environment variable and run the `configure` command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using `csh` or `tcsh`, you must issue commands somewhat differently:

```
shell> setenv CC gcc
shell> ./configure
```

1.3 Overview of the MySQL Database Management System

1.3.1 What is MySQL?

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.

The MySQL Web site (http://www.mysql.com/) provides the latest information about MySQL software.

- **MySQL is a database management system.**

  A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

- **MySQL databases are relational.**

  A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. The logical model, with objects such as databases, tables, views, rows, and columns, offers a flexible programming environment. You set up rules governing the relationships between different data fields, such as one-to-one, one-to-many, unique, required or optional, and “pointers” between different tables. The database enforces these rules, so that with a well-designed database, your application never sees inconsistent, duplicate, orphan, out-of-date, or missing data.

  The SQL part of “MySQL” stands for “Structured Query Language”. SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

  SQL is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” to mean the current version of the SQL Standard at any time.

- **MySQL software is Open Source.**

  Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), http://www.fsf.org/licenses/, to define what you may and may not do with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (http://www.mysql.com/company/legal/licensing/).

- **The MySQL Database Server is very fast, reliable, scalable, and easy to use.**

  If that is what you are looking for, you should give it a try. MySQL Server can run comfortably on a desktop or laptop, alongside your other applications, web servers, and so on, requiring little or no attention. If you dedicate an entire machine to MySQL, you can adjust the settings to take advantage of all the memory, CPU power, and I/O capacity available. MySQL can also scale up to clusters of machines, networked together.

  You can find a performance comparison of MySQL Server with other database managers on our benchmark page. See Section 7.1.3, “The MySQL Benchmark Suite”.

  MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years.

Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- **MySQL Server works in client/server or embedded systems.**

  The MySQL Database Software is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

  We also provide MySQL Server as an embedded multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage standalone product.

- **A large amount of contributed MySQL software is available.**

  MySQL Server has a practical set of features developed in close cooperation with our users. It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we do not mind if you pronounce it as “my sequel” or in some other localized way.

1.3.2 The Main Features of MySQL

This section describes some of the important characteristics of the MySQL Database Software. See also Section 1.4, “MySQL Development History”. In most respects, the roadmap applies to all versions of MySQL. For information about features as they are introduced into MySQL on a series-specific basis, see the “In a Nutshell” section of the appropriate Manual:

- MySQL 5.6: MySQL 5.6 in a Nutshell
- MySQL 5.5: MySQL 5.5 in a Nutshell
- MySQL 5.1: MySQL 5.1 in a Nutshell
- MySQL 5.0: MySQL 5.0 in a Nutshell

Internals and Portability:

- Written in C and C++.
- Tested with a broad range of different compilers.
- Works on many different platforms. See `http://www.mysql.com/support/supportedplatforms/database.html`.
- For portability, uses `CMake` in MySQL 5.5 and up. Previous series use GNU Automake, Autoconf, and Libtool.
- Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (http://developer.kde.org/~sewardj/).
- Uses multi-layered server design with independent modules.
- Designed to be fully multi-threaded using kernel threads, to easily use multiple CPUs if they are available.
- Provides transactional and nontransactional storage engines.

- Uses very fast B-tree disk tables (MyISAM) with index compression.
- Designed to make it relatively easy to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database.
- Uses a very fast thread-based memory allocation system.
- Executes very fast joins using an optimized nested-loop join.
- Implements in-memory hash tables, which are used as temporary tables.
- Implements SQL functions using a highly optimized class library that should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- Provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types:

- Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, FLOAT, DOUBLE, CHAR, VARCHAR, BINARY, VARBINARY, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, ENUM, and OpenGIS spatial types. See Chapter 10, *Data Types*.
- Fixed-length and variable-length string types.

Statements and Functions:

- Full operator and function support in the SELECT list and WHERE clause of queries. For example:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
    -> FROM citizen
    -> WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL GROUP BY and ORDER BY clauses. Support for group functions (COUNT() [882], AVG() [881], STD() [884], SUM() [884], MAX() [884], MIN() [884], and GROUP_CONCAT() [883]).
- Support for LEFT OUTER JOIN and RIGHT OUTER JOIN with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- Support for DELETE, INSERT, REPLACE, and UPDATE to return the number of rows that were changed (affected), or to return the number of rows matched instead by setting a flag when connecting to the server.
- Support for MySQL-specific SHOW statements that retrieve information about databases, storage engines, tables, and indexes. MySQL 5.0 adds support for the INFORMATION_SCHEMA database, implemented according to standard SQL.
- An EXPLAIN statement to show how the optimizer resolves a query.
- Independence of function names from table or column names. For example, ABS is a valid column name. The only restriction is that for a function call, no spaces are permitted between the function name and the "(" that follows it. See Section 8.3, "Reserved Words".
- You can refer to tables from different databases in the same statement.

Security:

- A privilege and password system that is very flexible and secure, and that enables host-based verification.
- Password security by encryption of all password traffic when you connect to a server.

Scalability and Limits:

- Support for large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 200,000 tables and about 5,000,000,000 rows.
- Support for up to 64 indexes per table (32 before MySQL 4.1.2). Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 767 bytes for `InnoDB` tables, or 1000 for `MyISAM`; before MySQL 4.1.2, the limit is 500 bytes. An index may use a prefix of a column for `CHAR`, `VARCHAR`, `BLOB`, or `TEXT` column types.

Connectivity:

- Clients can connect to MySQL Server using several protocols:
  - Clients can connect using TCP/IP sockets on any platform.
  - On Windows systems in the NT family (NT, 2000, XP, 2003, or Vista), clients can connect using named pipes if the server is started with the `--enable-named-pipe` [386] option. In MySQL 4.1 and higher, Windows servers also support shared-memory connections if started with the `--shared-memory` [392] option. Clients can connect through shared memory by using the `--protocol=memory` option.
  - On Unix systems, clients can connect using Unix domain socket files.
- MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, enabling MySQL clients to be written in many languages. See Chapter 17, *Connectors and APIs*.
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. Connector/ODBC source is available. All ODBC 2.5 functions are supported, as are many others. See MySQL Connector/ODBC Developer Guide.
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See MySQL Connector/J Developer Guide.
- MySQL Connector/Net enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/Net is a fully managed ADO.NET driver written in 100% pure C#. See MySQL Connector/Net Developer Guide.

Localization:

- The server can provide error messages to clients in many languages. See Section 9.3, “Setting the Error Message Language”.

- Full support for several different character sets, including `latin1` (cp1252), `german`, `big5`, `ujis`, and more. For example, the Scandinavian characters “å”, “ä” and “ö” are permitted in table and column names. Unicode support is available as of MySQL 4.1.
- All data is saved in the chosen character set.
- Sorting and comparisons are done according to the chosen character set and collation (using `latin1` and Swedish collation by default). It is possible to change this when the MySQL server is started. To see an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.
- As of MySQL 4.1, the server time zone can be changed dynamically, and individual clients can specify their own time zone. Section 9.7, “MySQL Server Time Zone Support”.

Clients and Tools:

- MySQL includes several client and utility programs. These include both command-line programs such as `mysqldump` and `mysqladmin`, and graphical programs such as MySQL Workbench.
- MySQL Server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the `mysqlcheck` client. MySQL also includes `myisamchk`, a very fast command-line utility for performing these operations on `MyISAM` tables. See Chapter 4, *MySQL Programs*.
- MySQL programs can be invoked with the `--help` or `-?` option to obtain online assistance.

1.3.3 History of MySQL

We started out with the intention of using the `mSQL` database system to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was designed to enable third-party code that was written for use with `mSQL` to be ported easily for use with MySQL.

MySQL is named after co-founder Monty Widenius's daughter, My.

The name of the MySQL Dolphin (our logo) is “Sakila,” which was chosen from a huge list of names suggested by users in our “Name the Dolphin” contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the feminine name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.4 MySQL Development History

This section describes the general MySQL development history, including major features implemented in or planned for various MySQL releases. The following sections provide information for each release series.

The current production release series is MySQL 5.1, which was declared stable for production use as of MySQL 5.1.30, released in November 2008. The previous production release series was MySQL 5.0, which was declared stable for production use as of MySQL 5.0.15, released in October 2005. “General Availability status” means that future 5.1 and 5.0 development is limited only to bugfixes. For the older MySQL 4.1, 4.0, and 3.23 series, only critical bugfixes are made.

Before upgrading from one release series to the next, please see the notes in Section 2.11.1, “Upgrading MySQL”.

The most requested features and the versions in which they were implemented are summarized in the following table.

Feature	MySQL Series
Unions	4.0
Subqueries	4.1
R-trees	4.1 (for the MyISAM storage engine)
Stored procedures and functions	5.0
Views	5.0
Cursors	5.0
XA transactions	5.0
Triggers	5.0 and 5.1
Event scheduler	5.1
Partitioning	5.1
Pluggable storage engine API	5.1
Plugin API	5.1
InnoDB Plugin	5.1
Row-based replication	5.1
Server log tables	5.1

1.5 MySQL 4.0 in a Nutshell

The following features were added in MySQL 4.0:

- Speed enhancements
  - MySQL 4.0 implemented a query cache that can give a major speed boost to applications with repetitive queries. See Section 7.5.3, “The MySQL Query Cache”.
  - MySQL 4.0 further increased the speed of MySQL Server in a number of areas, such as bulk INSERT statements, searching on packed indexes, full-text searching (using FULLTEXT indexes), and COUNT(DISTINCT) [882].
- InnoDB storage engine as standard
  - The InnoDB storage engine began to be offered as a standard feature of the MySQL server. This provided full support for ACID transactions, foreign keys with cascading UPDATE and DELETE, and row-level locking as standard features. See Section 13.2, “The InnoDB Storage Engine”.
- New functionality
  - The enhanced FULLTEXT search capabilities of MySQL Server 4.0 enabled FULLTEXT indexing of large text masses with both binary and natural-language searching logic. It became possible to customize minimal word length and define your own stop word lists in most human languages, enabling a broader class of applications to be built with MySQL Server. See Section 11.9, “Full-Text Search Functions”.
- Standards compliance, portability, and migration
  - MySQL Server added support for the UNION statement, a standard SQL feature.
  - Starting with version 4.0, MySQL runs natively on Novell NetWare 6.0 and higher. See Section 2.7, “Installing MySQL on NetWare”.

  - Features to simplify migration from other database systems to MySQL Server include `TRUNCATE TABLE` (as in Oracle) and `identity` [414] as a synonym for automatically incremented keys (as in Sybase).

- Internationalization

  - German-speaking users should note that MySQL 4.0 added support for a new character set, `latin1_de`, which ensures that words with umlauts are sorted in the same order as in German telephone books.

- Usability enhancements

  - As of version 4.0, most `mysqld` parameters (startup options) can be set without taking down the server. This is a convenient feature for database administrators. See Section 12.4.4, "`SET` Syntax".

  - Multiple-table `DELETE` and `UPDATE` statements were added.

  - On Windows, symbolic link handling at the database level was enabled by default. On Unix, the `MyISAM` storage engine added support for symbolic linking at the table level (and not just the database level as before).

  - The addition of the `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` [872] functions made it possible to find out the number of rows a `SELECT` query that includes a `LIMIT` clause would have returned without that clause.

- The Embedded MySQL Server

  The embedded server library added in this release can easily be used to create standalone and embedded applications. The embedded server provides an alternative to using MySQL in a client/server environment.

  The `libmysqld` embedded server library made MySQL Server suitable for a wider range of applications. Using this library, developers can embed MySQL Server into various applications and electronics devices, where the end user has no knowledge of there actually being an underlying database. Embedded MySQL Server is ideal for use in Internet appliances, public kiosks, turnkey hardware/software combination units, high performance Internet servers, self-contained databases distributed on CD-ROM, and so on.

  The embedded MySQL library uses the same interface as the normal client library. See Section 17.5, "libmysqld, the Embedded MySQL Server Library". Embedded MySQL is available under the same dual-licensing model as the MySQL Server; see http://www.mysql.com/company/legal/licensing/ for more information.

  On Windows, there are two different libraries, as shown in the following table.

Library Name	Library Type
`libmysqld.lib`	Dynamic library for threaded applications.
`mysqldemb.lib`	Static library for not threaded applications.

The news section of this manual includes a more in-depth list of MySQL 4.0 features. See Section C.2, "Changes in Release 4.0.x (Lifecycle Support Ended)".

1.6 MySQL 4.1 in a Nutshell

The following features were added in MySQL 4.1.

- **Support for subqueries and derived tables**:
  - A “subquery” is a `SELECT` statement nested within another statement. A “derived table” (an unnamed view) is a subquery in the `FROM` clause of another statement. See Section 12.2.8, “Subquery Syntax”.
- **Speed enhancements**:
  - Faster binary client/server protocol with support for prepared statements and parameter binding. See Section 17.6.7, “C API Prepared Statements”.
  - `BTREE` indexing is supported for `HEAP` tables, significantly improving response time for nonexact searches.
- **Added functionality**:
  - `CREATE TABLE tbl_name2 LIKE tbl_name1` enables you to create, with a single statement, a new table with a structure exactly like that of an existing table.
  - The `MyISAM` storage engine added support for OpenGIS spatial types for storing geographical data. See Chapter 16, *Spatial Extensions*.
  - Support was added for replication over SSL connections.
  - Support for a number of additional storage engines was implemented in the MySQL 4.1 release series:
    - The `EXAMPLE` storage engine is a “stub” engine that serves as an example in the MySQL source code for writing new storage engines, and is primarily of interest to developers. See Section 13.6, “The `EXAMPLE` Storage Engine”.
    - `NDBCLUSTER` is the storage engine used by MySQL Cluster to implement tables that are partitioned over many computers. See Chapter 15, *MySQL Cluster*.
    - The `ARCHIVE` storage engine is used for storing large amounts of data without indexes in a very small footprint. See Section 13.7, “The `ARCHIVE` Storage Engine”.
    - The `CSV` storage engine stores data in text files using comma-separated values format. See Section 13.8, “The `CSV` Storage Engine”.
    - The `BLACKHOLE` storage engine accepts but does not store data, and always returns an empty result set. It is for use primarily in replication. See Section 13.9, “The `BLACKHOLE` Storage Engine”.

Note

These engine were implemented at different points in the development of MySQL 4.1. Please see the indicated sections for particulars in each case.

- **Standards compliance, portability, and migration**:
  - The enhanced client/server protocol available beginning with MySQL 4.1.1 provides the ability to pass multiple warnings to the client, rather than only a single result, making it much easier to track problems that occur in operations such as bulk data loading.
  - `SHOW WARNINGS` shows warnings for the last command. See Section 12.4.5.26, “`SHOW WARNINGS` Syntax”.
- **Internationalization and Localization**:

- To support applications that require the use of local languages, the MySQL software added extensive Unicode support through the `utf8` and `ucs2` character sets.
- Definition of character sets by column, table, and database. This enables a high degree of flexibility in application design, particularly for multi-language Web sites. See Section 9.1, “Character Set Support”.
- Per-connection time zones support, enabling individual clients to select their own time zones when necessary.

- **Usability enhancements**:
  - The addition of a server-based `HELP` statement that can be used to get help information for SQL statements. This information is always applicable to the particular server version being used. Because this information is available by issuing an SQL statement, any client can access it. For example, the `help` command of the `mysql` command-line client has been modified to have this capability.
  - The improved client/server protocol permits multiple statements to be issued with a single call, and for returning multiple result sets. See Section 17.6.15, “C API Support for Multiple Statement Execution”.
  - The syntax `INSERT ... ON DUPLICATE KEY UPDATE ...` was implemented. This enables you to update an existing row if the insert would have caused a duplicate value for a primary or unique index. See Section 12.2.4, “`INSERT` Syntax”.
  - The aggregate function `GROUP_CONCAT()` [883], added the capability to concatenate column values from grouped rows into a single result string. See Section 11.15, “Functions and Modifiers for Use with `GROUP BY` Clauses”.

The News section of this manual includes a more in-depth list of MySQL 4.1 features. See Section C.1, “Changes in Release 4.1.x (Lifecycle Support Ended)”.

1.7 MySQL Information Sources

This section lists sources of additional information that you may find helpful, such as the MySQL mailing lists and user forums, and Internet Relay Chat.

1.7.1 MySQL Mailing Lists

This section introduces the MySQL mailing lists and provides guidelines as to how the lists should be used. When you subscribe to a mailing list, you receive all postings to the list as email messages. You can also send your own questions and answers to the list.

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit http://lists.mysql.com/. For most of them, you can select the regular version of the list where you get individual messages, or a digest version where you get one large message per day.

Please *do not* send messages about subscribing or unsubscribing to any of the mailing lists, because such messages are distributed automatically to thousands of other users.

Your local site may have many subscribers to a MySQL mailing list. If so, the site may have a local mailing list, so that messages sent from `lists.mysql.com` to your site are propagated to the local list. In such cases, please contact your system administrator to be added to or dropped from the local MySQL list.

To have traffic for a mailing list go to a separate mailbox in your mail program, set up a filter based on the message headers. You can use either the `List-ID:` or `Delivered-To:` headers to identify list messages.

The MySQL mailing lists are as follows:

- `announce`

  The list for announcements of new versions of MySQL and related programs. This is a low-volume list to which all MySQL users should subscribe.

- `mysql`

  The main list for general MySQL discussion. Please note that some topics are better discussed on the more-specialized lists. If you post to the wrong list, you may not get an answer.

- `bugs`

  The list for people who want to stay informed about issues reported since the last release of MySQL or who want to be actively involved in the process of bug hunting and fixing. See Section 1.8, “How to Report Bugs or Problems”.

- `internals`

  The list for people who work on the MySQL code. This is also the forum for discussions on MySQL development and for posting patches.

- `mysqldoc`

  The list for people who work on the MySQL documentation.

- `benchmarks`

  The list for anyone interested in performance issues. Discussions concentrate on database performance (not limited to MySQL), but also include broader categories such as performance of the kernel, file system, disk system, and so on.

- `packagers`

  The list for discussions on packaging and distributing MySQL. This is the forum used by distribution maintainers to exchange ideas on packaging MySQL and on ensuring that MySQL looks and feels as similar as possible on all supported platforms and operating systems.

- `java`

  The list for discussions about the MySQL server and Java. It is mostly used to discuss JDBC drivers such as MySQL Connector/J.

- `win32`

  The list for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, XP, and 2003.

- `myodbc`

  The list for all topics concerning connecting to the MySQL server with ODBC.

- `gui-tools`

  The list for all topics concerning MySQL graphical user interface tools such as MySQL Workbench.

- `cluster`

The list for discussion of MySQL Cluster.

- `dotnet`

  The list for discussion of the MySQL server and the .NET platform. It is mostly related to MySQL Connector/Net.

- `plusplus`

  The list for all topics concerning programming with the C++ API for MySQL.

- `perl`

  The list for all topics concerning Perl support for MySQL with `DBD::mysql`.

If you're unable to get an answer to your questions from a MySQL mailing list or forum, one option is to purchase support from Oracle. This puts you in direct contact with MySQL developers.

The following MySQL mailing lists are in languages other than English. These lists are not operated by Oracle.

- `<mysql-france-subscribe@yahoogroups.com>`

  A French mailing list.

- `<list@tinc.net>`

  A Korean mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

- `<mysql-de-request@lists.4t2.com>`

  A German mailing list. To subscribe, email `subscribe mysql-de your@email.address` to this list. You can find information about this mailing list at http://www.4t2.com/mysql/.

- `<mysql-br-request@listas.linkway.com.br>`

  A Portuguese mailing list. To subscribe, email `subscribe mysql-br your@email.address` to this list.

- `<mysql-alta@elistas.net>`

  A Spanish mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

1.7.1.1 Guidelines for Using the Mailing Lists

Please do not post mail messages from your browser with HTML mode turned on. Many users do not read mail with a browser.

When you answer a question sent to a mailing list, if you consider your answer to have broad interest, you may want to post it to the list instead of replying directly to the individual who asked. Try to make your answer general enough that people other than the original poster may benefit from it. When you post to the list, please make sure that your answer is not a duplication of a previous answer.

Try to summarize the essential part of the question in your reply. Do not feel obliged to quote the entire original message.

When answers are sent to you individually and not to the mailing list, it is considered good etiquette to summarize the answers and send the summary to the mailing list so that others may have the benefit of responses you received that helped you solve your problem.

1.7.2 MySQL Community Support at the MySQL Forums

The forums at http://forums.mysql.com are an important community resource. Many forums are available, grouped into these general categories:

- Migration
- MySQL Usage
- MySQL Connectors
- Programming Languages
- Tools
- 3rd-Party Applications
- Storage Engines
- MySQL Technology
- SQL Standards
- Business

1.7.3 MySQL Community Support on Internet Relay Chat (IRC)

In addition to the various MySQL mailing lists and forums, you can find experienced community people on Internet Relay Chat (IRC). These are the best networks/channels currently known to us:

freenode (see http://www.freenode.net/ for servers)

- `#mysql` is primarily for MySQL questions, but other database and general SQL questions are welcome. Questions about PHP, Perl, or C in combination with MySQL are also common.

If you are looking for IRC client software to connect to an IRC network, take a look at `xChat` (http://www.xchat.org/). X-Chat (GPL licensed) is available for Unix as well as for Windows platforms (a free Windows build of X-Chat is available at http://www.silverex.org/download/).

1.7.4 MySQL Enterprise

Oracle offers technical support in the form of MySQL Enterprise. For organizations that rely on the MySQL DBMS for business-critical production applications, MySQL Enterprise is a commercial subscription offering which includes:

- MySQL Enterprise Server
- MySQL Enterprise Monitor
- Monthly Rapid Updates and Quarterly Service Packs
- MySQL Knowledge Base
- 24x7 Technical and Consultative Support

MySQL Enterprise is available in multiple tiers, giving you the flexibility to choose the level of service that best matches your needs. For more information, see MySQL Enterprise.

1.8 How to Report Bugs or Problems

Before posting a bug report about a problem, please try to verify that it is a bug and that it has not been reported already:

- Start by searching the MySQL online manual at http://dev.mysql.com/doc/. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. In addition, the release notes accompanying the manual can be particularly useful since it is quite possible that a newer version contains a solution to your problem. The release notes are available at the location just given for the manual.
- If you get a parse error for an SQL statement, please check your syntax closely. If you cannot find something wrong with it, it is extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement.

  If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.
- For solutions to some common problems, see Section B.5, “Problems and Common Errors”.
- Search the bugs database at http://bugs.mysql.com/ to see whether the bug has been reported and fixed.
- Search the MySQL mailing list archives at http://lists.mysql.com/. See Section 1.7.1, “MySQL Mailing Lists”.
- You can also use http://www.mysql.com/search/ to search all the Web pages (including the manual) that are located at the MySQL Web site.

If you cannot find an answer in the manual, the bugs database, or the mailing list archives, check with your local MySQL expert. If you still cannot find an answer to your question, please use the following guidelines for reporting the bug.

The normal way to report bugs is to visit http://bugs.mysql.com/, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

Bugs posted in the bugs database at http://bugs.mysql.com/ that are corrected for a given release are noted in the release notes.

If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support at http://support.oracle.com/.

To discuss problems with other users, you can use one of the MySQL mailing lists. Section 1.7.1, “MySQL Mailing Lists”.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you do not waste your time doing things that may not help us much or at all. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test < script_file` on your test case or by running the shell or Perl script that you include in the bug report. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See Section 18.4, “Porting to Other Systems”.

Remember that it is possible for us to respond to a report containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details do not matter. A good principle to follow is that if you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you used. Note that every compiling problem should be regarded as a bug and reported accordingly.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) It is best to copy and paste the entire error message into your report. You should never try to reproduce the message from memory.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See How to Report Connector/ODBC Problems or Bugs.

If your report includes long query output lines from test cases that you run with the `mysql` command-line tool, you can make the output more readable by using the `--vertical` [265] option or the `\G` statement terminator. The `EXPLAIN SELECT` example later in this section demonstrates the use of `\G`.

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.0.19). You can find out which version you are running by executing `mysqladmin version`. The `mysqladmin` program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the “Help/About Windows” menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.

- If `mysqld` died, you should also report the statement that crashed `mysqld`. You can usually get this information by running `mysqld` with query logging enabled, and then looking in the log after `mysqld` crashes. See Section 18.4, “Porting to Other Systems”.

- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE db_name.tbl_name` statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.

- The SQL mode in effect when the problem occurred can be significant, so please report the value of the `sql_mode` [429] system variable. For stored procedure, stored function, and trigger objects, the relevant `sql_mode` [429] value is the one in effect when the object was created. For a stored procedure or function, the `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION` statement shows the relevant SQL mode, or you can query `INFORMATION_SCHEMA` for the information:

  ```
  SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.ROUTINES;
  ```

  For triggers, you can use this statement:

  ```
  SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.TRIGGERS;
  ```

- For performance-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT ...`, and at least the number of rows that the `SELECT` statement produces. You should also include the output from `SHOW CREATE TABLE tbl_name` for each table that is involved. The more information you provide about your situation, the more likely it is that someone can help you.

  The following is an example of a very good bug report. The statements are run using the `mysql` command-line tool. Note the use of the `\G` statement terminator for statements that would otherwise provide very long output lines that are difficult to read.

  ```
  mysql> SHOW VARIABLES;
  mysql> SHOW COLUMNS FROM ...\G
         <output from SHOW COLUMNS>
  mysql> EXPLAIN SELECT ...\G
         <output from EXPLAIN>
  mysql> FLUSH STATUS;
  mysql> SELECT ...;
         <A short version of the output from SELECT,
         including the time taken to run the query>
  mysql> SHOW STATUS;
         <output from SHOW STATUS>
  ```

- If a bug or problem occurs while running `mysqld`, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should upload it to be attached to the bug report.

  If you cannot provide a script, you should at least include the output from `mysqladmin variables extended-status processlist` in your report to provide some information on how your system is performing.

- If you cannot produce a test case with only a few rows, or if the test table is too big to be included in the bug report (more than 10 rows), you should dump your tables using `mysqldump` and create a `README` file that describes your problem. Create a compressed archive of your files using `tar` and `gzip` or `zip`.

After you initiate a bug report for our bugs database at http://bugs.mysql.com/, click the Files tab in the bug report for instructions on uploading the archive to the bugs database.

- If you believe that the MySQL server produces a strange result from a statement, include not only the result, but also your opinion of what the result should be, and an explanation describing the basis for your opinion.

- When you provide an example of the problem, it is better to use the table names, variable names, and so forth that exist in your actual situation than to come up with new names. The problem could be related to the name of a table or variable. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. If you have data that you do not want to be visible to others in the bug report, you can upload it using the Files tab as previously described. If the information is really top secret and you do not want to show it even to us, go ahead and provide an example using other names, but please regard this as the last choice.

- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the `mysqld` server, as well as the options that you use to run any MySQL client programs. The options to programs such as `mysqld` and `mysql`, and to the `configure` script, are often key to resolving problems and are very relevant. It is never a bad idea to include them. If your problem involves a program written in a language such as Perl or PHP, please include the language processor's version number, as well as the version for any modules that the program uses. For example, if you have a Perl script that uses the `DBI` and `DBD::mysql` modules, include the version numbers for Perl, `DBI`, and `DBD::mysql`.

- If your question is related to the privilege system, please include the output of `mysqlaccess`, the output of `mysqladmin reload`, and all the error messages you get when trying to connect. When you test your privileges, you should first run `mysqlaccess`. After this, execute `mysqladmin reload version` and try to connect with the program that gives you trouble. `mysqlaccess` can be found in the `bin` directory under your MySQL installation directory.

- If you have a patch for a bug, do include it. But do not assume that the patch is all we need, or that we can use it, if you do not provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all. If so, we cannot use it.

  If we cannot verify the exact purpose of the patch, we will not use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch will not work, it may be useless.

- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team cannot guess such things without first using a debugger to determine the real cause of a bug.

- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.

- If your data appears corrupt or you get errors when you access a particular table, first check your tables with `CHECK TABLE`. If that statement reports any errors:

  - The `InnoDB` crash recovery mechanism handles cleanup when the server is restarted after being killed, so in typical operation there is no need to “repair” tables. If you encounter an error with `InnoDB` tables, restart the server and see whether the problem persists, or whether the error affected only cached data in memory. If data is corrupted on disk, consider restarting with the `innodb_force_recovery` [1069] option enabled so that you can dump the affected tables.

  - For non-transactional tables, try to repair them with `REPAIR TABLE` or with `myisamchk`. See Chapter 5, *MySQL Server Administration*.

  If you are running Windows, please verify the value of `lower_case_table_names` [418] using the `SHOW VARIABLES LIKE 'lower_case_table_names'` statement. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in Section 8.2.2, “Identifier Case Sensitivity”.

- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See Section 5.3.1, “The Error Log”. Please include any relevant information from this file in your bug report. Normally `mysqld` should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of `mysqld` dying, it is much easier for us to provide you with a fix for the problem. See Section B.5.1, “How to Determine What Is Causing a Problem”.
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See Section 2.1.2, “Choosing Which MySQL Distribution to Install”.

1.9 MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some of the differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, “SQL:2003” refers to the standard released in 2003, and “SQL:2008” refers to the most recent version of the standard, released in 2008. We use the phrase “the SQL standard” or “standard SQL” to mean the current version of the SQL Standard at any time.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The `HANDLER` interface is an example of this strategy. See Section 12.2.3, “`HANDLER` Syntax”.

We continue to support transactional and nontransactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

Currently, we are not targeting real-time support, although MySQL replication capabilities offer significant functionality.

In MySQL 4.1.2 in later, high-availability database clustering is supported by the `NDBCLUSTER` storage engine. See Chapter 15, *MySQL Cluster*.

XML support is to be implemented in a future version of the database server.

1.9.1 What Standards MySQL Follows

Our aim is to support the full ANSI/ISO SQL standard, but without making concessions to speed and quality of the code.

ODBC levels 0 to 3.51.

1.9.2 Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differentially for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

SQL modes control aspects of server operation such as what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting `mysqld` with the `--sql-mode="mode_value"` [394] option. Beginning with MySQL 4.1, you can also change the mode at runtime by setting the `sql_mode` [429] system variable with a `SET [GLOBAL|SESSION] sql_mode='mode_value'` statement.

For more information on setting the SQL mode, see Section 5.1.6, “Server SQL Modes”.

1.9.3 Running MySQL in ANSI Mode

You can tell `mysqld` to run in ANSI mode with the `--ansi` [384] startup option. Running the server in ANSI mode is the same as starting it with the following options:

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

As of MySQL 4.1.1, you can achieve the same effect at runtime by executing these two statements:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'ANSI';
```

You can see that setting the `sql_mode` [429] system variable to `'ANSI'` enables all SQL mode options that are relevant for ANSI mode as follows:

```
mysql> SET GLOBAL sql_mode='ANSI';
mysql> SELECT @@global.sql_mode;
        -> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

Running the server in ANSI mode with `--ansi` [384] is not quite the same as setting the SQL mode to `'ANSI'`. The `--ansi` [384] option affects the SQL mode and also sets the transaction isolation level. Setting the SQL mode to `'ANSI'` has no effect on the isolation level.

See Section 5.1.2, “Server Command Options”, and Section 1.9.2, “Selecting SQL Modes”.

1.9.4 MySQL Extensions to Standard SQL

MySQL Server supports some extensions that you probably won't find in other SQL DBMSs. Be warned that if you use them, your code won't be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the "`!`" character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

  MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to file names in the database directory. This has a few implications:

  - Database and table names are case sensitive in MySQL Server on operating systems that have case-sensitive file names (such as most Unix systems). See Section 8.2.2, "Identifier Case Sensitivity".
  - You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the `MyISAM` or `ISAM` storage engines. For example, it is possible to rename a `MyISAM` table by renaming the `.MYD`, `.MYI`, and `.frm` files to which the table corresponds. (Nevertheless, it is preferable to use `RENAME TABLE` or `ALTER TABLE ... RENAME` and let the server rename the files.)

  Database and table names cannot contain path name separator characters ("`/`", "`\`").

- General language syntax
  - By default, strings can be enclosed by either "`"`" or "`'`", not just by "`'`". (If the `ANSI_QUOTES` [458] SQL mode is enabled, strings can be enclosed only by "`'`" and the server interprets strings enclosed by "`"`" as identifiers.)
  - "`\`" is the escape character in strings.
  - In SQL statements, you can access tables from different databases with the *`db_name.tbl_name`* syntax. Some SQL servers provide the same functionality but call this `User space`. MySQL Server doesn't support tablespaces such as used in statements like this: `CREATE TABLE ralph.my_table ... IN my_tablespace`.
- SQL statement syntax
  - The `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
  - The `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE` statements. See Section 12.1.3, "`CREATE DATABASE` Syntax", Section 12.1.6, "`DROP DATABASE` Syntax", and Section 12.1.1, "`ALTER DATABASE` Syntax".
  - The `DO` statement.
  - `EXPLAIN SELECT` to obtain a description of how tables are processed by the query optimizer.
  - The `FLUSH` and `RESET` statements.

- The `SET` statement. See Section 12.4.4, “`SET` Syntax”.
- The `SHOW` statement. See Section 12.4.5, “`SHOW` Syntax”.
- Use of `LOAD DATA INFILE`. In many cases, this syntax is compatible with Oracle's `LOAD DATA INFILE`. See Section 12.2.5, “`LOAD DATA INFILE` Syntax”.
- Use of `RENAME TABLE`. See Section 12.1.9, “`RENAME TABLE` Syntax”.
- Use of `REPLACE` instead of `DELETE` plus `INSERT`. See Section 12.2.6, “`REPLACE` Syntax”.
- Use of `CHANGE col_name`, `DROP col_name`, or `DROP INDEX`, `IGNORE` or `RENAME` in `ALTER TABLE` statements. Use of multiple `ADD`, `ALTER`, `DROP`, or `CHANGE` clauses in an `ALTER TABLE` statement. See Section 12.1.2, “`ALTER TABLE` Syntax”.
- Use of index names, indexes on a prefix of a column, and use of `INDEX` or `KEY` in `CREATE TABLE` statements. See Section 12.1.5, “`CREATE TABLE` Syntax”.
- Use of `TEMPORARY` or `IF NOT EXISTS` with `CREATE TABLE`.
- Use of `IF EXISTS` with `DROP TABLE` and `DROP DATABASE`.
- The capability of dropping multiple tables with a single `DROP TABLE` statement.
- The `ORDER BY` and `LIMIT` clauses of the `UPDATE` and `DELETE` statements.
- `INSERT INTO tbl_name SET col_name = ...` syntax.
- The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
- The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE`, and `UPDATE` statements.
- Use of `INTO OUTFILE` or `INTO DUMPFILE` in `SELECT` statements. See Section 12.2.7, “`SELECT` Syntax”.
- Options such as `STRAIGHT_JOIN` or `SQL_SMALL_RESULT` in `SELECT` statements.
- You don't need to name all selected columns in the `GROUP BY` clause. This gives better performance for some very specific, but quite normal queries. See Section 11.15, “Functions and Modifiers for Use with `GROUP BY` Clauses”.
- You can specify `ASC` and `DESC` with `GROUP BY`, not just with `ORDER BY`.
- The ability to set variables in a statement with the `:=` assignment operator:

```
mysql> SELECT @a:=SUM(total),@b:=COUNT(*),@a/@b AS avg
    -> FROM test_table;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

- Data types
  - The `MEDIUMINT`, `SET`, and `ENUM` data types, and the various `BLOB` and `TEXT` data types.
  - The `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, and `ZEROFILL` data type attributes.
- Functions and operators

- To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
- MySQL Server understands the || [787] and && [787] operators to mean logical OR and AND, as in the C programming language. In MySQL Server, || [787] and OR [787] are synonyms, as are && [787] and AND [787]. Because of this nice syntax, MySQL Server doesn't support the standard SQL || [787] operator for string concatenation; use CONCAT() [795] instead. Because CONCAT() [795] takes any number of arguments, it is easy to convert use of the || [787] operator to MySQL Server.
- Use of COUNT(DISTINCT *value_list*) [882] where *value_list* has more than one element.
- String comparisons are case-insensitive by default, with sort ordering determined by the collation of the current character set, which is latin1 (cp1252 West European) by default. If you don't like this, you should declare your columns with the BINARY attribute or use the BINARY cast, which causes comparisons to be done using the underlying character code values rather then a lexical ordering.
- The % [821] operator is a synonym for MOD() [821]. That is, *N* % *M* is equivalent to MOD(*N*,*M*) [821]. % [821] is supported for C programmers and for compatibility with PostgreSQL.
- The = [782], <> [782], <= [782], < [782], >= [782], > [783], << [863], >> [863], <=> [782], AND [787], OR [787], or LIKE [804] operators may be used in expressions in the output column list (to the left of the FROM) in SELECT statements. For example:

```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```

- The LAST_INSERT_ID() [874] function returns the most recent AUTO_INCREMENT value. See Section 11.13, “Information Functions”.
- LIKE [804] is permitted on numeric values.
- The REGEXP [808] and NOT REGEXP [808] extended regular expression operators.
- CONCAT() [795] or CHAR() [794] with one argument or more than two arguments. (In MySQL Server, these functions can take a variable number of arguments.)
- The BIT_COUNT() [863], CASE [790], ELT() [795], FROM_DAYS() [834], FORMAT() [796], IF() [790], PASSWORD() [868], ENCRYPT() [867], MD5() [868], ENCODE() [867], DECODE() [866], PERIOD_ADD() [837], PERIOD_DIFF() [838], TO_DAYS() [841], and WEEKDAY() [844] functions.
- Use of TRIM() [803] to trim substrings. Standard SQL supports removal of single characters only.
- The GROUP BY functions STD() [884], BIT_OR() [882], BIT_AND() [882], BIT_XOR() [882], and GROUP_CONCAT() [883]. See Section 11.15, “Functions and Modifiers for Use with GROUP BY Clauses”.

1.9.5 MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- For VARCHAR columns, trailing spaces are removed when the value is stored. See Section B.5.8, “Known Issues in MySQL”.

- In some cases, `CHAR` columns are silently converted to `VARCHAR` columns when you define a table or alter its structure. See Section 12.1.5.2, “Silent Column Specification Changes”.
- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a `REVOKE` statement to revoke privileges for a table. For more information, see Section 12.4.1.3, “`REVOKE` Syntax”.
- The `CAST()` [859] function does not support cast to `REAL` or `BIGINT`. See Section 11.10, “Cast Functions and Operators”.

1.9.5.1 Subquery Support

MySQL 4.1 and up supports subqueries and derived tables. A “subquery” is a `SELECT` statement nested within another statement. A “derived table” (an unnamed view) is a subquery in the `FROM` clause of another statement. See Section 12.2.8, “Subquery Syntax”.

For MySQL versions older than 4.1, most subqueries can be rewritten using joins or other methods. See Section 12.2.8.11, “Rewriting Subqueries as Joins for Earlier MySQL Versions”, for examples that show how to do this.

1.9.5.2 `SELECT INTO TABLE`

MySQL Server doesn't support the `SELECT ... INTO TABLE` Sybase SQL extension. Instead, MySQL Server supports the `INSERT INTO ... SELECT` standard SQL syntax, which is basically the same thing. See Section 12.2.4.1, “`INSERT ... SELECT` Syntax”. For example:

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use `SELECT ... INTO OUTFILE` or `CREATE TABLE ... SELECT`.

1.9.5.3 `UPDATE`

If you access a column from the table to be updated in an expression, `UPDATE` uses the current value of the column. The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.9.5.4 Transactions and Atomic Operations

MySQL Server (version 3.23-max and all versions 4.0 and above) supports transactions with the `InnoDB` and `BDB` transactional storage engines. `InnoDB` provides *full* ACID compliance. MySQL Cluster is also a transaction-safe storage engine. See Chapter 13, *Storage Engines*. For information about `InnoDB` differences from standard SQL with regard to treatment of transaction errors, see Section 13.2.13, “`InnoDB` Error Handling”.

The other nontransactional storage engines in MySQL Server (such as `MyISAM`) follow a different paradigm for data integrity called “atomic operations.” In transactional terms, `MyISAM` tables effectively always operate in `autocommit = 1` [406] mode. Atomic operations often offer comparable integrity with higher performance.

Because MySQL Server supports both paradigms, you can decide whether your applications are best served by the speed of atomic operations or the use of transactional features. This choice can be made on a per-table basis.

As noted, the tradeoff for transactional versus nontransactional storage engines lies mostly in performance. Transactional tables have significantly higher memory and disk space requirements, and more CPU overhead. On the other hand, transactional storage engines such as `InnoDB` also offer many significant features. MySQL Server's modular design enables the concurrent use of different storage engines to suit different requirements and deliver optimum performance in all situations.

But how do you use the features of MySQL Server to maintain rigorous integrity even with the nontransactional `MyISAM` tables, and how do these features compare with the transactional storage engines?

- If your applications are written in a way that is dependent on being able to call `ROLLBACK` rather than `COMMIT` in critical situations, transactions are more convenient. Transactions also ensure that unfinished updates or corrupting activities are not committed to the database; the server is given the opportunity to do an automatic rollback and your database is saved.

  If you use nontransactional tables, MySQL Server in almost all cases enables you to resolve potential problems by including simple checks before updates and by running simple scripts that check the databases for inconsistencies and automatically repair or warn if such an inconsistency occurs. You can normally fix tables perfectly with no data integrity loss just by using the MySQL log or even adding one extra log.

- More often than not, critical transactional updates can be rewritten to be atomic. Generally speaking, all integrity problems that transactions solve can be done with `LOCK TABLES` or atomic updates, ensuring that there are no automatic aborts from the server, which is a common problem with transactional database systems.

- To be safe with MySQL Server, regardless of whether you use transactional tables, you only need to have backups and have binary logging turned on. When that is true, you can recover from any situation that you could with any other transactional database system. It is always good to have backups, regardless of which database system you use.

The transactional paradigm has its advantages and disadvantages. Many users and application developers depend on the ease with which they can code around problems where an abort appears to be necessary, or is necessary. However, even if you are new to the atomic operations paradigm, or more familiar with transactions, do consider the speed benefit that nontransactional tables can offer on the order of three to five times the speed of the fastest and most optimally tuned transactional tables.

In situations where integrity is of highest importance, MySQL Server offers transaction-level reliability and integrity even for nontransactional tables. If you lock tables with `LOCK TABLES`, all updates stall until integrity checks are made. If you obtain a `READ LOCAL` lock (as opposed to a write lock) for a table that enables concurrent inserts at the end of the table, reads are permitted, as are inserts by other clients. The newly inserted records are not be seen by the client that has the read lock until it releases the lock. With `INSERT DELAYED`, you can write inserts that go into a local queue until the locks are released, without having the client wait for the insert to complete. See Section 7.6.3, “Concurrent Inserts”, and Section 12.2.4.2, “`INSERT DELAYED` Syntax”.

“Atomic,” in the sense that we mean it, is nothing magical. It only means that you can be sure that while each specific update is running, no other user can interfere with it, and there can never be an automatic rollback (which can happen with transactional tables if you are not very careful). MySQL Server also guarantees that there are no dirty reads.

Following are some techniques for working with nontransactional tables:

- Loops that need transactions normally can be coded with the help of `LOCK TABLES`, and you don't need cursors to update records on the fly.
- To avoid using `ROLLBACK`, you can employ the following strategy:

  1. Use `LOCK TABLES` to lock all the tables you want to access.
  2. Test the conditions that must be true before performing the update.
  3. Update if the conditions are satisfied.
  4. Use `UNLOCK TABLES` to release your locks.

  This is usually a much faster method than using transactions with possible rollbacks, although not always. The only situation this solution doesn't handle is when someone kills the threads in the middle of an update. In that case, all locks are released but some of the updates may not have been executed.
- You can also use functions to update records in a single operation. You can get a very efficient application by using the following techniques:

  - Modify columns relative to their current value.
  - Update only those columns that actually have changed.

  For example, when we are updating customer information, we update only the customer data that has changed and test only that none of the changed data, or data that depends on the changed data, has changed compared to the original row. The test for changed data is done with the `WHERE` clause in the `UPDATE` statement. If the record wasn't updated, we give the client a message: “Some of the data you have changed has been changed by another user.” Then we show the old row versus the new row in a window so that the user can decide which version of the customer record to use.

  This gives us something that is similar to column locking but is actually even better because we only update some of the columns, using values that are relative to their current values. This means that typical `UPDATE` statements look something like these:

  ```
  UPDATE tablename SET pay_back=pay_back+125;

  UPDATE customer
    SET
      customer_date='current_date',
      address='new address',
      phone='new phone',
      money_owed_to_us=money_owed_to_us-125
    WHERE
      customer_id=id AND address='old address' AND phone='old phone';
  ```

  This is very efficient and works even if another client has changed the values in the `pay_back` or `money_owed_to_us` columns.
- In many cases, users have wanted `LOCK TABLES` or `ROLLBACK` for the purpose of managing unique identifiers. This can be handled much more efficiently without locking or rolling back by using an `AUTO_INCREMENT` column and either the `LAST_INSERT_ID()` [874] SQL function or the `mysql_insert_id()` C API function. See Section 11.13, “Information Functions”, and Section 17.6.6.35, “`mysql_insert_id()`”.

  You can generally code around the need for row-level locking. Some situations really do need it, and `InnoDB` tables support row-level locking. Otherwise, with `MyISAM` tables, you can use a flag column in the table and do something like the following:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID;
```

MySQL returns `1` for the number of affected rows if the row was found and `row_flag` wasn't `1` in the original row. You can think of this as though MySQL Server changed the preceding statement to:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
```

1.9.5.5 Stored Routines and Triggers

Stored procedures and functions are implemented beginning with MySQL 5.0.

Basic trigger functionality is implemented beginning with MySQL 5.0.2, with further development planned for MySQL 5.1.

1.9.5.6 Foreign Keys

The `InnoDB` storage engine supports checking of foreign key constraints, including `CASCADE`, `ON DELETE`, and `ON UPDATE`. See Section 13.2.5.4, “`FOREIGN KEY` Constraints”.

For storage engines other than `InnoDB`, MySQL Server parses the `FOREIGN KEY` syntax in `CREATE TABLE` statements, but does not use or store it. In the future, the implementation will be extended to store this information in the table specification file so that it may be retrieved by `mysqldump` and ODBC. At a later stage, foreign key constraints will be implemented for `MyISAM` tables as well.

Foreign key enforcement offers several benefits to database developers:

- Assuming proper design of the relationships, foreign key constraints make it more difficult for a programmer to introduce an inconsistency into the database.
- Centralized checking of constraints by the database server makes it unnecessary to perform these checks on the application side. This eliminates the possibility that different applications may not all check the constraints in the same way.
- Using cascading updates and deletes can simplify the application code.
- Properly designed foreign key rules aid in documenting relationships between tables.

Do keep in mind that these benefits come at the cost of additional overhead for the database server to perform the necessary checks. Additional checking by the server affects performance, which for some applications may be sufficiently undesirable as to be avoided if possible. (Some major commercial applications have coded the foreign key logic at the application level for this reason.)

MySQL gives database developers the choice of which approach to use. If you don't need foreign keys and want to avoid the overhead associated with enforcing referential integrity, you can choose another storage engine instead, such as `MyISAM`. (For example, the `MyISAM` storage engine offers very fast performance for applications that perform only `INSERT` and `SELECT` operations. In this case, the table has no holes in the middle and the inserts can be performed concurrently with retrievals. See Section 7.6.3, “Concurrent Inserts”.)

If you choose not to take advantage of referential integrity checks, keep the following considerations in mind:

- In the absence of server-side foreign key relationship checking, the application itself must handle relationship issues. For example, it must take care to insert rows into tables in the proper order, and to avoid creating orphaned child records. It must also be able to recover from errors that occur in the middle of multiple-record insert operations.

- If `ON DELETE` is the only referential integrity capability an application needs, you can achieve a similar effect as of MySQL Server 4.0 by using multiple-table `DELETE` statements to delete rows from many tables with a single statement. See Section 12.2.1, “`DELETE` Syntax”.
- A workaround for the lack of `ON DELETE` is to add the appropriate `DELETE` statements to your application when you delete records from a table that has a foreign key. In practice, this is often as quick as using foreign keys and is more portable.

Be aware that the use of foreign keys can sometimes lead to problems:

- Foreign key support addresses many referential integrity issues, but it is still necessary to design key relationships carefully to avoid circular rules or incorrect combinations of cascading deletes.
- It is not uncommon for a DBA to create a topology of relationships that makes it difficult to restore individual tables from a backup. (MySQL alleviates this difficulty by enabling you to temporarily disable foreign key checks when reloading a table that depends on other tables. See Section 13.2.5.4, “`FOREIGN KEY` Constraints”. As of MySQL 4.1.1, `mysqldump` generates dump files that take advantage of this capability automatically when they are reloaded.)

Foreign keys in SQL are used to check and enforce referential integrity, not to join tables. If you want to get results from multiple tables from a `SELECT` statement, you do this by performing a join between them:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

See Section 12.2.7.1, “`JOIN` Syntax”, and Section 3.6.6, “Using Foreign Keys”.

The `FOREIGN KEY` syntax without `ON DELETE ...` is often used by ODBC applications to produce automatic `WHERE` clauses.

1.9.5.7 Views

Views (including updatable views) are implemented beginning with MySQL Server 5.0.1.

Views are useful for enabling users to access a set of relations (tables) as if it were a single table, and limiting their access to just that. Views can also be used to restrict access to rows (a subset of a particular table). For access control to columns, you can also use the sophisticated privilege system in MySQL Server. See Section 5.5, “The MySQL Access Privilege System”.

In designing an implementation of views, our ambitious goal, as much as is possible within the confines of SQL, has been full compliance with “Codd's Rule #6” for relational database systems: “All views that are theoretically updatable, should in practice also be updatable.”

1.9.5.8 '`--`' as the Start of a Comment

Standard SQL uses the C syntax `/* this is a comment */` for comments, and MySQL Server supports this syntax as well. MySQL also support extensions to this syntax that enable MySQL-specific SQL to be embedded in the comment, as described in Section 8.6, “Comment Syntax”.

Standard SQL uses “`--`” as a start-comment sequence. MySQL Server uses “`#`” as the start comment character. MySQL Server 3.23.3 and up also supports a variant of the “`--`” comment style. That is, the “`--`” start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for `payment`:

```
UPDATE account SET credit=credit-payment
```

Consider about what happens if `payment` has a negative value such as `-1`:

```
UPDATE account SET credit=credit--1
```

`credit--1` is a legal expression in SQL, but "`--`" is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all. This illustrates that permitting comments to start with "`--`" can have serious consequences.

Using our implementation requires a space following the "`--`" for it to be recognized as a start-comment sequence in MySQL Server 3.23.3 and newer. Therefore, `credit--1` is safe to use.

Another safe feature is that the `mysql` command-line client ignores lines that start with "`--`".

The following information is relevant only if you are running a MySQL version earlier than 3.23.3:

If you have an SQL script in a text file that contains "`--`" comments, you should use the `replace` utility as follows to convert the comments to use "`#`" characters before executing the script:

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \
        | mysql db_name
```

That is safer than executing the script in the usual way:

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

You can also edit the script file "in place" to change the "`--`" comments to "`#`" comments:

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Change them back with this command:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

See Section 4.8.2, "`replace` — A String-Replacement Utility".

1.9.6 How MySQL Deals with Constraints

MySQL enables you to work both with transactional tables that permit rollback and with nontransactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other DBMSs. We must handle the case when you have inserted or updated a lot of rows in a nontransactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce illegal values to the closest legal values.

The following sections describe how MySQL Server handles different types of constraints.

1.9.6.1 PRIMARY KEY and UNIQUE Index Constraints

Normally, errors occurs for data-change statements (such as INSERT or UPDATE) that would violate primary-key, unique-key, or foreign-key constraints. If you are using a transactional storage engine such as InnoDB, MySQL automatically rolls back the statement. If you are using a nontransactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

MySQL supports an IGNORE keyword for INSERT, UPDATE, and so forth. If you use it, MySQL ignores primary-key or unique-key violations and continues processing with the next row. See the section for the statement that you are using (Section 12.2.4, “INSERT Syntax”, Section 12.2.9, “UPDATE Syntax”, and so forth).

You can get information about the number of rows actually inserted or updated with the mysql_info() C API function. In MySQL 4.1 and up, you also can use the SHOW WARNINGS statement. See Section 17.6.6.33, “mysql_info()”, and Section 12.4.5.26, “SHOW WARNINGS Syntax”.

Currently, only InnoDB tables support foreign keys. See Section 13.2.5.4, “FOREIGN KEY Constraints”.

1.9.6.2 Constraints on Invalid Data

Through version 4.1, MySQL is forgiving of illegal or improper data values and coerces them to legal values for data entry. When you insert an “incorrect” value into a column, such as a NULL into a NOT NULL column or a too-large numeric value into a numeric column, MySQL sets the column to the “best possible value” instead of producing an error. The following rules describe in more detail how this works:

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that doesn't start with a number into a numeric column, MySQL Server stores 0.
- Invalid values for ENUM and SET columns are handled as described in Section 1.9.6.3, “ENUM and SET Constraints”.
- MySQL enables you to store certain incorrect date values into DATE and DATETIME columns (such as '2000-02-31' or '2000-02-00'). The idea is that it is not the job of the SQL server to validate dates. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value '0000-00-00' is stored in the column instead.
- If you try to store NULL into a column that doesn't take NULL values, an error occurs for single-row INSERT statements. For multiple-row INSERT statements or for INSERT INTO ... SELECT statements, MySQL Server stores the implicit default value for the column data type. In general, this is 0 for numeric types, the empty string ('') for string types, and the “zero” value for date and time types. Implicit default values are discussed in Section 10.1.4, “Data Type Default Values”.
- If an INSERT statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit DEFAULT clause. If the definition has no such DEFAULT clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in

this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it is better to “do the best you can” and then continue as if nothing happened.

1.9.6.3 `ENUM` and `SET` Constraints

`ENUM` and `SET` columns provide an efficient way to define columns that can contain only a given set of values. See Section 10.4.4, “The `ENUM` Type”, and Section 10.4.5, “The `SET` Type”. However, in MySQL 4.1 and earlier, `ENUM` and `SET` columns do not provide true constraints on entry of invalid data:

- `ENUM` columns always have a default value. If you specify no default value, then it is `NULL` for columns that can have `NULL`, otherwise it is the first enumeration value in the column definition.
- If you insert an incorrect value into an `ENUM` column or if you force a value into an `ENUM` column with `IGNORE`, it is set to the reserved enumeration value of `0`, which is displayed as an empty string in string context.
- If you insert an incorrect value into a `SET` column, the incorrect value is ignored. For example, if the column can contain the values `'a'`, `'b'`, and `'c'`, an attempt to assign `'a,x,b,y'` results in a value of `'a,b'`.

1.10 Credits

The following sections list developers, contributors, and supporters that have helped to make MySQL what it is today.

1.10.1 Contributors to MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize those who have made contributions of one kind or another to the `MySQL distribution`. Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola `<qwerg@mbox.vol.it>` or `<qwerg@tin.it>`

  The initial port to Win32/NT.

- Per Eric Olsson

  For constructive criticism and real testing of the dynamic record format.

- Irena Pancirov `<irena@mail.yacc.it>`

  Win32 port with Borland compiler. `mysqlshutdown.exe` and `mysqlwatch.exe`.

- David J. Hughes

  For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with `mSQL`, but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. `mysqladmin` and `mysql` client are programs that were largely influenced by their `mSQL` counterparts. We have put a lot of effort into making the MySQL syntax a superset of `mSQL`. Many of the API's ideas are borrowed from `mSQL` to make it easy to port free `mSQL` programs to the MySQL API. The MySQL software doesn't contain any code from `mSQL`. Two files in the distribution (`client/insert_test.c` and `client/select_test.c`) are based on the corresponding (noncopyrighted) files in the `mSQL` distribution, but are modified as examples showing the changes necessary to convert code from `mSQL` to MySQL Server. (`mSQL` is copyrighted David J. Hughes.)

- Patrick Lynch

For helping us acquire http://www.mysql.com/.

- Fred Lindberg

  For setting up qmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko <igor@frog.kiev.ua>

  mysqldump (previously msqldump, but ported and enhanced by Monty).

- Yuri Dario

  For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

  Author of mysqlhotcopy.

- Zarko Mocnik <zarko.mocnik@dem.si>

  Sorting for Slovenian language.

- "TAMITO" <tommy@valley.ne.jp>

  The _MB character set macros and the ujis and sjis character sets.

- Joshua Chamas <joshua@chamas.com>

  Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier <Yves.Carlier@rug.ac.be>

  mysqlaccess, a program to show the access rights for a user.

- Rhys Jones <rhys@wales.com> (And GWE Technologies Limited)

  For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>

  Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper <pixel@organic.com>

  For setting up a searchable mailing list archive at his site.

- Rick Mehalick <Rick_Mehalick@i-o.com>

  For xmysql, a graphical X client for MySQL Server.

- Doug Sisk <sisk@wix.com>

  For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. <axeld@vial.ethz.ch>

  For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive <toni@readysoft.es>

  For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

  For providing RPM versions for MySQL 3.21.

- David Sacerdote <davids@secnet.com>

  Ideas for secure checking of DNS host names.

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

  Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

  A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

  Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

  `FROM_UNIXTIME()` [834] time formatting, `ENCRYPT()` [867] functions, and `bison` advisor. Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

  Ported (and extended) the benchmark suite to `DBI`/`DBD`. Have been of great help with `crash-me` and running benchmarks. Some new date functions. The `mysql_setpermission` script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

  User-defined functions (UDFs); `CREATE FUNCTION` and `DROP FUNCTION`.

- Andreas F. Bobak <bobak@relog.ch>

  The `AGGREGATE` extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

  Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

  The `libmysql.dll` library.

- James Pereria <jpereira@iafrica.com>

  Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

  Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

- Martin Ramsch <m.ramsch@computer.org>

Examples in the MySQL Tutorial.

- Steve Harvey

  For making `mysqlaccess` more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

  http://www.pspl.co.in/konark/. Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

  Configure updates for Tru64, large file support and better TCP wrappers support.

- John Birrell

  Emulation of `pthread_mutex()` for OS/2.

- Benjamin Pflugmann

  Extended `MERGE` tables to handle `INSERTS`. Active member on the MySQL mailing lists.

- Jocelyn Fournier

  Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).

- Marc Liyanage

  Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X packages.

- Robert Rutherford

  Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

  Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataullah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

- Google Inc.

  We wish to recognize Google Inc. for contributions to the MySQL distribution: Mark Callaghan's SMP Performance patches and other patches.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, `<jehamby@lightside>`, `<psmith@BayNetworks.com>`, `<duane@connect.com.au>`, Ted Deppner `<ted@psyber.com>`, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch <dkoch@amcity.com>

  Irix setup.

- Luuk de Boer <luuk@wxs.nl>

  Benchmark questions.

- Tim Sailer <tps@users.buoy.com>

  DBD::mysql questions.

- Boyd Lynn Gerber <gerberb@zenez.com>

  SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

  xmysql-related questions and basic installation questions.

- Zeev Suraski <bourbon@netvision.net.il>

  Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch <frankie@citel.upc.es>

  General questions.

- Jonathan J Smith <jsmith@wtp.net>

  Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

  Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

  Is flexible and can handle Linux and perhaps HP-UX.

- John Lyon <jlyon@imag.net>

  Questions about installing MySQL on Linux systems, using either .rpm files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

  Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

  ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

  DBD, Linux, some SQL syntax questions.

1.10.2 Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

  Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

  Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

  For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

  First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (http://mysql.hitstar.com/) versions were based. Personal home page at linuxdb.yeah.net.

- Jay Flaherty <fty@mediapulse.com>

  Big parts of the Perl DBI/DBD section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

  Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

  French error messages.

- Petr Snajdr, <snajdr@pvt.net>

  Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

  Polish error messages.

- Miguel Angel Fernandez Roiz

  Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>

  Norwegian error messages and testing of MySQL 3.21.xx.

- Timur I. Bakeyev <root@timur.tatarstan.ru>

  Russian error messages.

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

  Italian error messages.

- Dirk Munzinger <dirk@trinity.saar.de>

German error messages.

- Billik Stefan <billik@sun.uniag.sk>

  Slovak error messages.

- Stefan Saroiu <tzoompy@cs.washington.edu>

  Romanian error messages.

- Peter Feher

  Hungarian error messages.

- Roberto M. Serqueira

  Portuguese error messages.

- Carsten H. Pedersen

  Danish error messages.

- Arjen Lentz

  Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

1.10.3 Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We cannot list every possible package here because the list would then be way to hard to maintain. For other packages, please refer to the software portal at http://solutions.mysql.com/software/.

- Tim Bunce, Alligator Descartes

  For the `DBD` (Perl) interface.

- Andreas Koenig <a.koenig@mind.de>

  For the Perl interface for MySQL Server.

- Jochen Wiedmann <wiedmann@neckar-alb.de>

  For maintaining the Perl `DBD::mysql` module.

- Eugene Chan <eugene@acenet.com.sg>

  For porting PHP for MySQL Server.

- Georg Richter

  MySQL 4.1 testing and bug hunting. New PHP 5.0 `mysqli` extension (API) for use with MySQL 4.1 and up.

- Giovanni Maruzzelli <maruzz@matrice.it>

  For porting iODBC (Unix ODBC).

- Xavier Leroy <Xavier.Leroy@inria.fr>

  The author of LinuxThreads (used by the MySQL Server on Linux).

1.10.4 Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation

  From whom we got an excellent compiler (`gcc`), an excellent debugger (`gdb` and the `libc` library (from which we have borrowed `strto.c` to get some code working in Linux).

- Free Software Foundation & The XEmacs development team

  For a really great editor/environment.

- Julian Seward

  Author of `valgrind`, an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.

- Dorothea Lütkehaus and Andreas Zeller

  For `DDD` (The Data Display Debugger) which is an excellent graphical front end to `gdb`).

1.10.5 Supporters of MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize the following companies, which helped us finance the development of the `MySQL server`, such as by paying us for developing a new feature or giving us hardware for development of the `MySQL server`.

- VA Linux / Andover.net

  Funded replication.

- NuSphere

  Editing of the MySQL manual.

- Stork Design studio

  The MySQL Web site in use between 1998-2000.

- Intel

  Contributed to development on Windows and Linux platforms.

- Compaq

  Contributed to Development on Linux/Alpha.

- SWSoft

  Development on the embedded `mysqld` version.

- FutureQuest

  The `--skip-show-database` [394] option.

Chapter 2 Installing and Upgrading MySQL

Table of Contents

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This chapter describes how to obtain and install MySQL. A summary of the procedure follows and later sections provide the details. If you plan to upgrade an existing version of MySQL to a newer version rather than install MySQL for the first time, see Section 2.11.1, “Upgrading MySQL”, for information about upgrade procedures and about issues that you should consider before upgrading.

1. **Determine whether MySQL runs and is supported on your platform.** Please note that not all platforms are equally suitable for running MySQL, and that not all platforms on which MySQL is known to run are officially supported by Oracle Corporation. See Section 2.1.1, “Operating Systems On Which MySQL Is Known To Run”, for details.

2. **Choose which distribution to install.** Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. We also provide public access to our current source tree for those who want to see our most recent developments and help us test new code. To determine which version and type of distribution you should use, see Section 2.1.2, “Choosing Which MySQL Distribution to Install”.

3. **Download the distribution that you want to install.** For instructions, see Section 2.1.3, “How to Get MySQL”. To verify the integrity of the distribution, use the instructions in Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

4. **Install the distribution.** To install MySQL from a binary distribution, use the instructions in Section 2.2, “Standard MySQL Installation from a Binary Distribution”. To install MySQL from a source distribution or from the current development source tree, use the instructions in Section 2.9, “Installing MySQL from Source”.

   If you encounter installation difficulties, see Section 2.12, “Operating System-Specific Notes”, for information on solving problems for particular platforms.

5. **Perform any necessary postinstallation setup.** After installing MySQL, read Section 2.10, “Postinstallation Setup and Testing”. This section contains important information about making sure the MySQL server is working properly. It also describes how to secure the initial MySQL user accounts, *which have no passwords* until you assign passwords. The section applies whether you install MySQL using a binary or source distribution.

6. If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See Section 2.14, “Perl Installation Notes”.

2.1 General Installation Guidance

Before installing MySQL, you should do the following:

1. Determine whether MySQL runs on your platform.
2. Choose a distribution to install.
3. Download the distribution and verify its integrity.

This section contains the information necessary to carry out these steps. After doing so, you can use the instructions in later sections of the chapter to install the distribution that you choose.

2.1.1 Operating Systems On Which MySQL Is Known To Run

This section lists the operating systems on which MySQL is known to run.

Important

Oracle Corporation does not necessarily provide official support for all the platforms listed in this section. For information about those platforms that are officially supported, see http://www.mysql.com/support/supportedplatforms/database.html on the MySQL Web site.

We use GNU Autoconf, so it is possible to port MySQL to all modern systems that have a C++ compiler and a working implementation of POSIX threads. (Thread support is needed for the server. To compile only the client code, the only requirement is a C++ compiler.)

MySQL has been reported to compile successfully on the following combinations of operating system and thread package.

- AIX 4.x, 5.x with native threads. See Section 2.12.5.3, “IBM-AIX notes”.
- Amiga.
- FreeBSD 5.x and up with native threads.
- HP-UX 11.x with the native threads. See Section 2.12.5.2, “HP-UX Version 11.x Notes”.
- Linux, builds on all fairly recent Linux distributions with `glibc` 2.3. See Section 2.12.1, “Linux Notes”.
- Mac OS X. See Section 2.5, “Installing MySQL on Mac OS X”.
- NetBSD 1.3/1.4 Intel and NetBSD 1.3 Alpha. See Section 2.12.4.2, “NetBSD Notes”.
- Novell NetWare 6.0 and 6.5. See Section 2.7, “Installing MySQL on NetWare”.
- OpenBSD 2.5 and with native threads. OpenBSD earlier than 2.5 with the MIT-pthreads package. See Section 2.12.4.3, “OpenBSD 2.5 Notes”.
- SCO OpenServer 5.0.X with a recent port of the FSU Pthreads package. See Section 2.12.5.8, “SCO UNIX and OpenServer 5.0.x Notes”.
- SCO Openserver 6.0.x. See Section 2.12.5.9, “SCO OpenServer 6.0.x Notes”.
- SCO UnixWare 7.1.x. See Section 2.12.5.10, “SCO UnixWare 7.1.x and OpenUNIX 8.0.0 Notes”.
- SGI Irix 6.x with native threads. See Section 2.12.5.7, “SGI Irix Notes”.
- Solaris 2.5 and above with native threads on SPARC and x86. See Section 2.12.3, “Solaris Notes”.

- Tru64 Unix. See Section 2.12.5.5, “Alpha-DEC-UNIX Notes (Tru64)”.
- Windows 9x, Me, NT, 2000, XP, and Windows Server 2003. See Section 2.3, “Installing MySQL on Microsoft Windows”.

MySQL has also been known to run on other systems in the past. See Section 2.12, “Operating System-Specific Notes”. Some porting effort might be required for current versions of MySQL on these systems.

Not all platforms are equally well-suited for running MySQL. How well a certain platform is suited for a high-load mission-critical MySQL server is determined by the following factors:

- General stability of the thread library. A platform may have an excellent reputation otherwise, but MySQL is only as stable as the thread library it calls, even if everything else is perfect.
- The capability of the kernel and the thread library to take advantage of symmetric multi-processor (SMP) systems. In other words, when a process creates a thread, it should be possible for that thread to run on a CPU different from the original process.
- The capability of the kernel and the thread library to run many threads that acquire and release a mutex over a short critical region frequently without excessive context switches. If the implementation of `pthread_mutex_lock()` is too anxious to yield CPU time, this hurts MySQL tremendously. If this issue is not taken care of, adding extra CPUs actually makes MySQL slower.
- General file system stability and performance.
- Table size. If your tables are large, performance is affected by the ability of the file system to deal with large files at all and to deal with them efficiently.
- Our level of expertise here at Oracle Corporation with the platform. If we know a platform well, we enable platform-specific optimizations and fixes at compile time. We can also provide advice on configuring your system optimally for MySQL.
- The amount of testing we have done internally for similar configurations.
- The number of users that have run MySQL successfully on the platform in similar configurations. If this number is high, the likelihood of encountering platform-specific surprises is much smaller.

2.1.2 Choosing Which MySQL Distribution to Install

When preparing to install MySQL, you should decide which version to use. MySQL development occurs in several release series, and you can pick the one that best fits your needs. After deciding which version to install, you can choose a distribution format. Releases are available in binary or source format.

2.1.2.1 Choosing Which Version of MySQL to Install

The first decision to make is whether you want to use a production (stable) release or a development release. In the MySQL development process, multiple release series co-exist, each at a different stage of maturity.

Production Releases

- MySQL 5.6: Latest General Availability (Production) release
- MySQL 5.5: Previous General Availability (Production) release
- MySQL 5.1: Older General Availability (Production) release
- MySQL 5.0: Older Production release nearing the end of the product lifecycle

MySQL 4.1, 4.0, and 3.23 are old releases that are no longer supported.

See http://www.mysql.com/about/legal/lifecycle/ for information about support policies and schedules.

Normally, if you are beginning to use MySQL for the first time or trying to port it to some system for which there is no binary distribution, use the most recent General Availability series listed in the preceding descriptions. All MySQL releases, even those from development series, are checked with the MySQL benchmarks and an extensive test suite before being issued.

If you are running an older system and want to upgrade, but do not want to take the chance of having a nonseamless upgrade, you should upgrade to the latest version in the same release series you are using (where only the last part of the version number is newer than yours). We have tried to fix only fatal bugs and make only small, relatively “safe” changes to that version.

If you want to use new features not present in the production release series, you can use a version from a development series. Be aware that development releases are not as stable as production releases.

We do not use a complete code freeze because this prevents us from making bugfixes and other fixes that must be done. We may add small things that should not affect anything that currently works in a production release. Naturally, relevant bugfixes from an earlier series propagate to later series.

If you want to use the very latest sources containing all current patches and bugfixes, you can use one of our source code repositories (see Section 2.9.2, “Installing MySQL from a Development Source Tree”). These are not “releases” as such, but are available as previews of the code on which future releases are to be based.

The naming scheme in MySQL 4.1 uses release names that consist of three numbers and a suffix; for example, `mysql-4.1.2-alpha`. The numbers within the release name are interpreted like this:

- The first number (4) is the major version and also describes the file format. All version 4 releases have the same file format.
- The second number (1) is the release level. Taken together, the major version and release level constitute the release series number.
- The third number (2) is the version number within the release series. This is incremented for each new release. Usually you want the latest version for the series you have chosen.

For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.

Release names also include a suffix to indicates the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- **alpha** indicates that the release is for preview purposes only. Known bugs should be documented in the News section (see Appendix C, *MySQL Release Notes*). Most alpha releases implement new commands and extensions. Active development that may involve major code changes can occur in an alpha release. However, we do conduct testing before issuing a release.
- **beta** indicates that the release is appropriate for use with new development. Within beta releases, the features and compatibility should remain consistent. However, beta releases may contain numerous and major unaddressed bugs.

  All APIs, externally visible structures, and columns for SQL statements will not change during future beta, release candidate, or production releases.
- **rc** indicates a Release Candidate. Release candidates are believed to be stable, having passed all of MySQL's internal testing, and with all known fatal runtime bugs fixed. However, the release has not been

in widespread use long enough to know for sure that all bugs have been identified. Only minor fixes are added. (A release candidate is what formerly was known as a gamma release.)

- If there is no suffix, it indicates that the release is a General Availability (GA) or Production release. GA releases are stable, having successfully passed through all earlier release stages and are believed to be reliable, free of serious bugs, and suitable for use in production systems. Only critical bugfixes are applied to the release.

All releases of MySQL are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Because the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

All releases have been tested at least with these tools:

- **An internal test suite.** The `mysql-test` directory contains an extensive set of test cases. We run these tests for every server binary. See Section 18.1.2, “The MySQL Test Suite”, for more information about this test suite.
- **The MySQL benchmark suite.** This suite runs a range of common queries. It is also a test to determine whether the latest batch of optimizations actually made the code faster. See Section 7.1.3, “The MySQL Benchmark Suite”.
- **The `crash-me` test.** This test tries to determine what features the database supports and what its capabilities and limitations are. See Section 7.1.3, “The MySQL Benchmark Suite”.

2.1.2.2 Choosing a Distribution Format

After choosing which version of MySQL to install, you should decide whether to use a binary distribution or a source distribution. In most cases, you should probably use a binary distribution, if one exists for your platform. Binary distributions are available in native format for many platforms, such as RPM packages for Linux, DMG packages for Mac OS X, and PKG packages for Solaris. Distributions are also available in more generic formats such as Zip archives or compressed `tar` files.

Reasons to choose a binary distribution include the following:

- Binary distributions generally are easier to install than source distributions.
- To satisfy different user requirements, we provide two different binary versions. One is compiled with the core feature set. The other (MySQL-Max) is compiled with an extended feature set. Both versions are compiled from the same source distribution. All native MySQL clients can connect to servers from either MySQL version.

  The extended MySQL binary distribution is identified by the `-max` suffix and is configured with the same options as `mysqld-max`. See Section 5.2, “The `mysqld-max` Extended MySQL Server”.

  For RPM distributions, if you want to use the `MySQL-Max` RPM, you must first install the standard `MySQL-server` RPM.

Under some circumstances, you may be better off installing MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ready to run at any installation location, but you might require even more flexibility to place MySQL components where you want.
- You want to configure `mysqld` to ensure that features are available that might not be included in the standard binary distributions. Here is a list of the most common extra options that you may want to use to ensure feature availability:

- `--with-berkeley-db` (not available on all platforms)
- `--with-raid`
- `--with-libwrap`
- `--with-named-z-libs` (this is done for some of the binaries)
- `--with-debug[=full]` [98]

- You want to configure `mysqld` without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You want to use the latest sources from one of the Bazaar repositories to have access to all current bugfixes. For example, if you have found a bug and reported it to the MySQL development team, the bugfix is committed to the source repository and you can access it there. The bugfix does not appear in a release until a release actually is issued.
- You want to read (or modify) the C and C++ code that makes up MySQL. For this purpose, you should get a source distribution, because the source code is always the ultimate manual.
- Source distributions contain more tests and examples than binary distributions.

2.1.2.3 How and When Updates Are Released

MySQL is evolving quite rapidly and we want to share new developments with other MySQL users. We try to produce a new release whenever we have new and useful features that others also seem to have a need for.

We also try to help users who request features that are easy to implement. We take note of what our licensed users want, and we especially take note of what our support customers want and try to help them in this regard.

No one is *required* to download a new release. The News section helps you determine whether the new release has something you really want. See Appendix C, *MySQL Release Notes*.

We use the following policy when updating MySQL:

- Releases are issued within each series. For each release, the last number in the version is one more than the previous release within the same series.
- Production (stable) releases are meant to appear about 1-2 times a year. However, if small bugs are found, a release with only bugfixes is issued.
- Working releases/bugfixes to old releases are meant to appear about every 4-8 weeks.
- Binary distributions for some platforms are made by us for major releases. Other people may make binary distributions for other systems, but probably less frequently.
- We make fixes available as soon as we have identified and corrected small or noncritical but annoying bugs. The fixes are available immediately from our public Bazaar repositories, and are included in the next release.
- If by any chance a fatal bug is found in a release, our policy is to fix it in a new release as soon as possible. (We would like other companies to do this, too!)

2.1.2.4 MySQL Binaries Compiled by Oracle Corporation

Oracle Corporation provides a set of binary distributions of MySQL. In addition to binaries provided in platform-specific package formats, we offer binary distributions for a number of platforms in the form of compressed `tar` files (`.tar.gz` files). See Section 2.2, “Standard MySQL Installation from a Binary Distribution”. For Windows distributions, see Section 2.3, “Installing MySQL on Microsoft Windows”.

If you want to compile MySQL from a source distribution, see Section 2.9, “Installing MySQL from Source”. To compile a debug version of MySQL, see Section 2.9.3, “MySQL Source-Configuration Options” for options that enable debugging.

2.1.3 How to Get MySQL

Check our downloads page at http://dev.mysql.com/downloads/ for information about the current version of MySQL and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see http://dev.mysql.com/downloads/mirrors.html. You can also find information there about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

To obtain the latest development source, see Section 2.9.2, “Installing MySQL from a Development Source Tree”.

2.1.4 Verifying Package Integrity Using MD5 Checksums or GnuPG

After you have downloaded the MySQL package that suits your needs and before you attempt to install it, you should make sure that it is intact and has not been tampered with. There are three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using `GnuPG`, the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site. If you repeatedly cannot successfully verify the integrity of the package, please notify us about such incidents, including the full package name and the download site you have been using, at `<webmaster@mysql.com>` or `<build@mysql.com>`. Do not report downloading problems using the bug-reporting system.

2.1.4.1 Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify with the following command, where *`package_name`* is the name of the package you downloaded:

```
shell> md5sum package_name
```

Example:

```
shell> md5sum mysql-standard-4.0.17-pc-linux-i686.tar.gz
60f5fe969d61c8f82e4f7f62657e1f06  mysql-standard-4.0.17-pc-linux-i686.tar.gz
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.

Note

Make sure to verify the checksum of the *archive file* (for example, the `.zip` or `.tar.gz` file) and not of the files that are contained inside of the archive.

Note that not all operating systems support the `md5sum` command. On some, it is simply called `md5`, and others do not ship it at all. On Linux, it is part of the **GNU Text Utilities** package, which is available for a wide range of platforms. You can download the source code from http://www.gnu.org/software/textutils/ as well. If you have OpenSSL installed, you can use the command `openssl md5 package_name` instead. A Windows implementation of the `md5` command line utility is available from http://www.fourmilab.ch/md5/. `winMd5Sum` is a graphical MD5 checking tool that can be obtained from http://www.nullriver.com/index/products/winmd5sum.

2.1.4.2 Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

Beginning with MySQL 4.0.10 (February 2003), we started signing downloadable packages with `GnuPG` (`GNU Privacy Guard`). `GnuPG` is an Open Source alternative to the very well-known `Pretty Good Privacy` (`PGP`) by Phil Zimmermann. See http://www.gnupg.org/ for more information about `GnuPG` and how to obtain and install it on your system. Most Linux distributions ship with `GnuPG` installed by default. For more information about `OpenPGP`, see http://www.openpgp.org/.

To verify the signature for a specific package, you first need to obtain a copy of our public GPG build key, which you can download from http://keyserver.pgp.com/. The key that you want to obtain is named `build@mysql.com`. Alternatively, you can cut and paste the key directly from the following text:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.5 (GNU/Linux)

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q7TXlTUUwgUGFj
a2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkgPGJ1aWxkQG15c3FsLmNv
bT6IXQQTEQIAHQULBwoDBAMVAwIDFgIBAheABQJLcC5lBQkQ8/JZAAoJEIxxjTtQ
cuH1oD4AoIcOQ4EoGsZvy06D0Ei5vcsWEy8dAJ4g46i3WEcdSWxMhcBSsPz65sh5
lohMBBMRAgAMBQI+PqPRBYMJZgC7AAoJEElQ4SqycpHyJOEAn1mxHijft00bKXvu
cSo/pECUmppiAJ41M9MRVj5VcdH/KN/KjRtW6tHFPYhMBBMRAgAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQ/lpEAoIhpp6BozKI8p6eaabzF5MlJH58pAKCu/ROofK8J
Eg2aLos+5zEYrB/LsrkCDQQ+PqMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWsEN/l
xaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb6O1JHffOo2HfLmRDRi
Rjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkbf4fm
Le11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8Ilbf5vSYHbuE5p
/1oIDznkg/p8kW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+Lwqq
a8CGrRfsOAJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaf
anFvwFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTbOW
I39EcyzgslzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTPtl0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKHt92
6s/ymfdf5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUOetdZ
Whe70YGNPw1yjWJT1IhMBBgRAgAMBQI+PqMdBQkJZgGAAAoJEIxxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295kl6NxYEuFApmr1+0uUq/SlsQ==
=Mski
```

```
-----END PGP PUBLIC KEY BLOCK-----
```

To import the build key into your personal public GPG keyring, use `gpg --import`. For example, if you have saved the key in a file named `mysql_pubkey.asc`, the import command looks like this:

```
shell> gpg --import mysql_pubkey.asc
gpg: key 5072E1F5: public key "MySQL Package signing key (www.mysql.com) <build@mysql.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg: no ultimately trusted keys found
```

You can also download the key from the public keyserver using the public key id, `5072E1F5`:

```
shell> gpg --recv-keys 5072E1F5
gpg: requesting key 5072E1F5 from hkp server subkeys.pgp.net
gpg: key 5072E1F5: "MySQL Package signing key (www.mysql.com) <build@mysql.com>" 2 new signatures
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:         new signatures: 2
```

If you want to import the key into your RPM configuration to validate RPM install packages, you should be able to import the key directly:

```
shell> rpm --import mysql_pubkey.asc
```

If you experience problems, try exporting the key from `gpg` and importing:

```
shell> gpg --export -a 5072e1f5 > 5072e1f5.asc
shell> rpm --import 5072e1f5.asc
```

Alternatively, `rpm` also supports loading the key directly from a URL, and you cas use this manual page:

```
shell> rpm --import http://dev.mysql.com/doc/refman/4.1/en/checking-gpg-signature.html
```

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension. For example:

File Type	File Name
Distribution file	`mysql-standard-4.0.17-pc-linux-i686.tar.gz`
Signature file	`mysql-standard-4.0.17-pc-linux-i686.tar.gz.asc`

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

Example:

```
shell> gpg --verify mysql-standard-4.0.17-pc-linux-i686.tar.gz.asc
gpg: Warning: using insecure memory!
gpg: Signature made Mon 03 Feb 2003 08:50:39 PM MET
using DSA key ID 5072E1F5
gpg: Good signature from
     "MySQL Package signing key (www.mysql.com) <build@mysql.com>"
```

The `Good signature` message indicates that everything is all right. You can ignore any `insecure memory` warning you might obtain.

See the GPG documentation for more information on how to work with public keys.

2.1.4.3 Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-4.1.25-0.glibc23.i386.rpm
MySQL-server-4.1.25-0.glibc23.i386.rpm: md5 gpg OK
```

Note

If you are using RPM 4.1 and it complains about `(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)`, even though you have imported the MySQL public build key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, it maintains its own keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key as described in Section 2.1.4.2, “Signature Checking Using GnuPG”. Then use `rpm --import` to import the key. For example, if you have saved the public key in a file named `mysql_pubkey.asc`, import it using this command:

```
shell> rpm --import mysql_pubkey.asc
```

If you need to obtain the MySQL public key, see Section 2.1.4.2, “Signature Checking Using GnuPG”.

2.1.5 Installation Layouts

This section describes the default layout of the directories created by installing binary or source distributions provided by Oracle Corporation. A distribution provided by another vendor might use a layout different from those shown here.

On Windows, the default installation directory is `C:\mysql`. With MySQL version 4.1.5 and higher, this has changed to `C:\Program Files\MySQL\MySQL Server 4.1`, where 4.1 is the major version of the installation. The folder has the following subdirectories:

Table 2.1 MySQL Installation Layout for Windows

Directory	Contents of Directory
`bin`	Client programs and the `mysqld` server
`data`	Log files, databases
`examples`	Example programs and scripts
`include`	Include (header) files
`lib`	Libraries
`scripts`	Utility scripts
`share`	Error message and character set files

Installations created from our Linux RPM distributions result in files under the following system directories:

Table 2.2 MySQL Installation Layout for Linux RPM

Directory	Contents of Directory
`/usr/bin`	Client programs and scripts
`/usr/sbin`	The `mysqld` server
`/var/lib/mysql`	Log files, databases
`/usr/share/info`	Manual in Info format
`/usr/share/man`	Unix manual pages
`/usr/include/mysql`	Include (header) files
`/usr/lib/mysql`	Libraries
`/usr/share/mysql`	Error message and character set files
`/usr/share/sql-bench`	Benchmarks

On Unix, a `tar` file binary distribution is installed by unpacking it at the installation location you choose (typically `/usr/local/mysql`) and creates the following directories in that location:

Table 2.3 MySQL Installation Layout for Generic Unix/Linux Binary Package

Directory	Contents of Directory
`bin`	Client programs and the `mysqld` server
`data`	Log files, databases
`docs`	Manual in Info format
`man`	Unix manual pages
`include`	Include (header) files
`lib`	Libraries
`scripts`	`mysql_install_db`
`share/mysql`	Error message and character set files
`sql-bench`	Benchmarks

By default, when you install MySQL after compiling it from a source distribution, the installation step installs files under `/usr/local`. Components are installed in the directories shown in the following table. To configure particular installation locations, use the options described at Section 2.9.3, “MySQL Source-Configuration Options”.

Table 2.4 MySQL Layout for Installation from Source

Directory	Contents of Directory
`bin`	Client programs and scripts
`include/mysql`	Include (header) files
`Docs`	Manual in Info format
`man`	Unix manual pages
`lib/mysql`	Libraries
`libexec`	The `mysqld` server
`share/mysql`	Error message and character set files
`sql-bench`	Benchmarks
`var`	Log files, databases

Within its installation directory, the layout of a source installation differs from that of a binary installation in the following ways:

- The `mysqld` server is installed in the `libexec` directory rather than in the `bin` directory.
- The data directory is `var` rather than `data`.
- `mysql_install_db` is installed in the `bin` directory rather than in the `scripts` directory.
- The header file and library directories are `include/mysql` and `lib/mysql` rather than `include` and `lib`.

To create your own binary installation from a compiled source distribution, execute the `scripts/make_binary_distribution` script from the top directory of the source distribution.

2.1.6 Compiler-Specific Build Characteristics

In some cases, the compiler used to build MySQL affects the features available for use. The notes in this section apply for binary distributions provided by Oracle Corporation or that you compile yourself from source.

`icc` (Intel C++ Compiler) Builds

A server built with `icc` has these characteristics:

- SSL support is not included.

2.2 Standard MySQL Installation from a Binary Distribution

The next several sections cover the installation of MySQL on platforms where we offer packages using the native packaging format of the respective platform. (This is also known as performing a “binary install.”) However, binary distributions of MySQL are available for many other platforms as well. See Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”, for generic installation instructions for these packages that apply to all platforms.

See Section 2.1, “General Installation Guidance”, for more information on what other binary distributions are available and how to obtain them.

2.3 Installing MySQL on Microsoft Windows

A native Windows distribution of MySQL has been available since version 3.21 and represents a sizable percentage of the daily downloads of MySQL. This section describes the process for installing MySQL on Windows.

MySQL 4.1.5 introduced a new installer for the Windows version of MySQL, combined with a new GUI Configuration Wizard. This combination automatically installs MySQL, creates an option file, starts the server, and secures the default user accounts.

Note

If you are upgrading MySQL from an existing installation older than MySQL 4.1.5, you must first perform the procedure described in Section 2.3.14, “Upgrading MySQL on Windows”.

To run MySQL on Windows, you need the following:

- A 32-bit Windows operating system such as 9x, Me, NT, 2000, XP, Vista, or Windows Server 2003.

  A Windows NT-based operating system (NT, 2000, XP, Vista, 2003) permits you to run the MySQL server as a service. The use of a Windows NT-based operating system is strongly recommended. See Section 2.3.11, “Starting MySQL as a Windows Service”.

  Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the `PATH` environment variable or accessing the `Service Control Manager`.

- TCP/IP protocol support.

- Enough space on the hard drive to unpack, install, and create the databases in accordance with your requirements (generally a minimum of 200 megabytes is recommended.)

For a list of limitations within the Windows version of MySQL, see Section D.3.3, “Windows Platform Limitations”.

There may also be other requirements, depending on how you plan to use MySQL:

- If you plan to connect to the MySQL server using ODBC, you need a Connector/ODBC driver. See Chapter 17, *Connectors and APIs*.

- If you need tables with a size larger than 4GB, install MySQL on an NTFS or newer file system. Do not forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See Section 12.1.5, “`CREATE TABLE` Syntax”.

MySQL for Windows is available in several distribution formats:

- Binary distributions are available that contain a setup program that installs everything you need so that you can start the server immediately. Another binary distribution format contains an archive that you simply unpack in the installation location and then configure yourself. For details, see Section 2.3.1, “Choosing An Installation Package”.

- The source distribution contains all the code and support files for building the executables using the Visual Studio 7.1 compiler system.

Generally speaking, you should use a binary distribution that includes an installer. It is simpler to use than the others, and you need no additional tools to get MySQL up and running. The installer for the Windows version of MySQL, combined with a GUI Configuration Wizard, automatically installs MySQL, creates an option file, starts the server, and secures the default user accounts.

The following section describes how to install MySQL on Windows using a binary distribution. To use an installation package that does not include an installer, follow the procedure described in Section 2.3.5, “Installing MySQL from a Noinstall Zip Archive”. To install using a source distribution, see Section 2.9.7, “Installing MySQL from Source on Windows”.

MySQL distributions for Windows can be downloaded from http://dev.mysql.com/downloads/. See Section 2.1.3, “How to Get MySQL”.

2.3.1 Choosing An Installation Package

Starting with MySQL version 4.1.5, there are three install packages to choose from when installing MySQL on Windows. The Packages are as follows:

- **The Essentials Package**: This package has a file name similar to `mysql-essential-4.1.13a-win32.msi` and contains the minimum set of files needed to install MySQL on Windows, including the

Configuration Wizard. This package does not include optional components such as the embedded server and benchmark suite.

- **The Complete Package**: This package has a file name similar to `mysql-4.1.13a-win32.zip` and contains all files needed for a complete Windows installation, including the Configuration Wizard. This package includes optional components such as the embedded server and benchmark suite.
- **The Noinstall Archive**: This package has a file name similar to `mysql-noinstall-4.1.13a-win32.zip` and contains all the files found in the Complete install package, with the exception of the Configuration Wizard. This package does not include an automated installer, and must be manually installed and configured.

The Essentials package is recommended for most users. It is provided as an `.msi` file for use with the Windows Installer. The Complete and Noinstall distributions are packaged as Zip archives. To use them, you must have a tool that can unpack `.zip` files.

Your choice of install package affects the installation process you must follow. If you choose to install either the Essentials or Complete install packages, see Section 2.3.2, “Installing MySQL with the Automated Installer”. If you choose to install MySQL from the Noinstall archive, see Section 2.3.5, “Installing MySQL from a Noinstall Zip Archive”.

2.3.2 Installing MySQL with the Automated Installer

Starting with MySQL 4.1.5, users can use the new MySQL Installation Wizard and MySQL Configuration Wizard to install MySQL on Windows. The MySQL Installation Wizard and MySQL Configuration Wizard are designed to install and configure MySQL in such a way that new users can immediately get started using MySQL.

The MySQL Installation Wizard and MySQL Configuration Wizard are available in the Essentials and Complete install packages. They are recommended for most standard MySQL installations. Exceptions include users who need to install multiple instances of MySQL on a single server host and advanced users who want complete control of server configuration.

If you are installing a version of MySQL prior to MySQL 4.1.5, please follow the instructions for installing MySQL from the Noinstall installation package. See Section 2.3.5, “Installing MySQL from a Noinstall Zip Archive”.

2.3.3 Using the MySQL Installation Wizard

2.3.3.1 Introduction to the Installation Wizard

MySQL Installation Wizard is an installer for the MySQL server that uses the latest installer technologies for Microsoft Windows. The MySQL Installation Wizard, in combination with the MySQL Configuration Wizard, enables a user to install and configure a MySQL server that is ready for use immediately after installation.

The MySQL Installation Wizard is the standard installer for all MySQL server distributions, version 4.1.5 and higher. Users of previous versions of MySQL need to shut down and remove their existing MySQL installations manually before installing MySQL with the MySQL Installation Wizard. See Section 2.3.3.7, “Upgrading MySQL with the Installation Wizard”, for more information on upgrading from a previous version.

Microsoft has included an improved version of their Microsoft Windows Installer (MSI) in the recent versions of Windows. MSI has become the de-facto standard for application installations on Windows 2000, Windows XP, and Windows Server 2003. The MySQL Installation Wizard makes use of this technology to provide a smoother and more flexible installation process.

The Microsoft Windows Installer Engine was updated with the release of Windows XP; those using a previous version of Windows can reference this Microsoft Knowledge Base article for information on upgrading to the latest version of the Windows Installer Engine.

In addition, Microsoft has introduced the WiX (Windows Installer XML) toolkit recently. This is the first highly acknowledged Open Source project from Microsoft. We have switched to WiX because it is an Open Source project and it enables us to handle the complete Windows installation process in a flexible manner using scripts.

Improving the MySQL Installation Wizard depends on the support and feedback of users like you. If you find that the MySQL Installation Wizard is lacking some feature important to you, or if you discover a bug, please report it in our bugs database using the instructions given in Section 1.8, “How to Report Bugs or Problems”.

2.3.3.2 Downloading and Starting the MySQL Installation Wizard

The MySQL installation packages can be downloaded from http://dev.mysql.com/downloads/. If the package you download is contained within a Zip archive, you need to extract the archive first.

Note

If you are installing on Windows Vista it is best to open a port before beginning the installation. To do this first ensure that you are logged in as an administrator, go to the `Control Panel`, and double-click the `Windows Firewall` icon. Choose the `Allow a program through Windows Firewall` option and click the Add port button. Enter `MySQL` into the **Name** text box and `3306` (or the port of your choice) into the **Port number** text box. Also ensure that the **TCP** protocol radio button is selected. If you wish, you can also limit access to the MySQL server by choosing the **Change scope** button. Confirm your choices by clicking the OK button. If you do not open a port prior to installation, you cannot configure the MySQL server immediately after installation. Additionally, when running the MySQL Installation Wizard on Windows Vista, ensure that you are logged in as a user with administrative rights.

The process for starting the wizard depends on the contents of the installation package you download. If there is a `setup.exe` file present, double-click it to start the installation process. If there is an `.msi` file present, double-click it to start the installation process.

2.3.3.3 Choosing an Install Type

There are three installation types available: **Typical**, **Complete**, and **Custom**.

The **Typical** installation type installs the MySQL server, the `mysql` command-line client, and the command-line utilities. The command-line clients and utilities include `mysqldump`, `myisamchk`, and several other tools to help you manage the MySQL server.

The **Complete** installation type installs all components included in the installation package. The full installation package includes components such as the embedded server library, the benchmark suite, support scripts, and documentation.

The **Custom** installation type gives you complete control over which packages you wish to install and the installation path that is used. See Section 2.3.3.4, “The Custom Install Dialog”, for more information on performing a custom install.

If you choose the **Typical** or **Complete** installation types and click the Next button, you advance to the confirmation screen to verify your choices and begin the installation. If you choose the **Custom** installation

type and click the Next button, you advance to the custom installation dialog, described in Section 2.3.3.4, “The Custom Install Dialog”.

2.3.3.4 The Custom Install Dialog

If you wish to change the installation path or the specific components that are installed by the MySQL Installation Wizard, choose the **Custom** installation type.

A tree view on the left side of the custom install dialog lists all available components. Components that are not installed have a red X icon; components that are installed have a gray icon. To change whether a component is installed, click that component's icon and choose a new option from the drop-down list that appears.

You can change the default installation path by clicking the Change... button to the right of the displayed installation path.

After choosing your installation components and installation path, click the Next button to advance to the confirmation dialog.

2.3.3.5 The Confirmation Dialog

Once you choose an installation type and optionally choose your installation components, you advance to the confirmation dialog. Your installation type and installation path are displayed for you to review.

To install MySQL if you are satisfied with your settings, click the Install button. To change your settings, click the Back button. To exit the MySQL Installation Wizard without installing MySQL, click the Cancel button.

After installation is complete, you have the option of registering with the MySQL Web site. Registration gives you access to post in the MySQL forums at forums.mysql.com, along with the ability to report bugs at bugs.mysql.com and to subscribe to our newsletter. The final screen of the installer provides a summary of the installation and gives you the option to launch the MySQL Configuration Wizard, which you can use to create a configuration file, install the MySQL service, and configure security settings.

2.3.3.6 Changes Made by MySQL Installation Wizard

Once you click the Install button, the MySQL Installation Wizard begins the installation process and makes certain changes to your system which are described in the sections that follow.

Changes to the Registry

The MySQL Installation Wizard creates one Windows registry key in a typical install situation, located in `HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB`.

The MySQL Installation Wizard creates a key named after the major version of the server that is being installed, such as `MySQL Server 4.1`. It contains two string values, `Location` and `Version`. The `Location` string contains the path to the installation directory. In a default installation it contains `C:\Program Files\MySQL\MySQL Server 4.1\`. The `Version` string contains the release number. For example, for an installation of MySQL Server 4.1.5, the key contains a value of `4.1.5`.

These registry keys are used to help external tools identify the installed location of the MySQL server, preventing a complete scan of the hard-disk to determine the installation path of the MySQL server. The registry keys are not required to run the server, and if you install MySQL using the `noinstall` Zip archive, the registry keys are not created.

Changes to the Start Menu

The MySQL Installation Wizard creates a new entry in the Windows Start menu under a common MySQL menu heading named after the major version of MySQL that you have installed. For example, if you install MySQL 4.1, the MySQL Installation Wizard creates a MySQL Server 4.1 section in the start menu.

The following entries are created within the new Start menu section:

- MySQL Command Line Client: This is a shortcut to the `mysql` command-line client and is configured to connect as the `root` user. The shortcut prompts for a `root` user password when you connect.
- MySQL Server Instance Config Wizard: This is a shortcut to the MySQL Configuration Wizard. Use this shortcut to configure a newly installed server, or to reconfigure an existing server.
- MySQL Documentation: This is a link to the MySQL server documentation that is stored locally in the MySQL server installation directory. This option is not available when the MySQL server is installed from the Essentials installation package.

Changes to the File System

The MySQL Installation Wizard by default installs the MySQL server to `C:\Program Files\MySQL\MySQL Server 4.1`, where `Program Files` is the default location for applications in your system, and `4.1` is the major version of your MySQL server. This is the new location for the MySQL server, replacing the former default location of `c:\mysql`.

By default, all MySQL applications are stored in a common directory at `C:\Program Files\MySQL`, where `Program Files` is the default location for applications in your Windows installation. A typical MySQL installation on a developer machine might look like this:

```
C:\Program Files\MySQL\MySQL Server 4.1
C:\Program Files\MySQL\MySQL Workbench 5.1 OSS
```

This approach makes it easier to manage and maintain all MySQL applications installed on a particular system.

2.3.3.7 Upgrading MySQL with the Installation Wizard

From MySQL version 4.1.5, the new MySQL Installation Wizard can perform server upgrades automatically using the upgrade capabilities of MSI. That means you do not need to remove a previous installation manually before installing a new release. The installer automatically shuts down and removes the previous MySQL service before installing the new version.

Automatic upgrades are available only when upgrading between installations that have the same major and minor version numbers. For example, you can upgrade automatically from MySQL 4.1.5 to MySQL 4.1.6, but not from MySQL 4.1 to MySQL 5.0.

If you are upgrading MySQL version 4.1.4 or earlier to version 4.1.5 or later, you must first manually shut down and remove the older installation before upgrading. Be sure to back up your databases before performing such an upgrade, so that you can restore the databases after the upgrade is completed. It is always recommended that you back up your data before performing any upgrades.

See Section 2.3.14, “Upgrading MySQL on Windows”.

2.3.4 Using the Configuration Wizard

2.3.4.1 Introduction to the Configuration Wizard

The MySQL Configuration Wizard helps automate the process of configuring your server under Windows. The MySQL Configuration Wizard creates a custom `my.ini` file by asking you a series of questions and then applying your responses to a template to generate a `my.ini` file that is tuned to your installation.

The MySQL Configuration Wizard is included with the MySQL server starting with MySQL version 4.1.5, but is designed to work with MySQL servers versions 4.1 and higher. The MySQL Configuration Wizard is currently available for Windows users only.

The MySQL Configuration Wizard is to a large extent the result of feedback that we have received from many users over a period of several years. However, if you find that it lacks some feature important to you, please report it in our bugs database using the instructions given in Section 1.8, “How to Report Bugs or Problems”.

2.3.4.2 Starting the MySQL Configuration Wizard

The MySQL Configuration Wizard is typically launched from the MySQL Installation Wizard, as the MySQL Installation Wizard exits. You can also launch the MySQL Configuration Wizard by clicking the MySQL Server Instance Config Wizard entry in the MySQL section of the Windows Start menu.

Alternatively, you can navigate to the `bin` directory of your MySQL installation and launch the `MySQLInstanceConfig.exe` file directly.

Note

If you chose not to open a port prior to installing MySQL on Windows Vista, you can choose to use the MySQL Server Configuration Wizard after installation. However, you must open a port in the Windows Firewall. To do this see the instructions given in Section 2.3.3.2, “Downloading and Starting the MySQL Installation Wizard”. Rather than opening a port, you also have the option of adding MySQL as a program that bypasses the Windows Firewall. One or the other option is sufficient—you need not do both. Additionally, when running the MySQL Server Configuration Wizard on Windows Vista ensure that you are logged in as a user with administrative rights.

2.3.4.3 Choosing a Maintenance Option

If the MySQL Configuration Wizard detects an existing `my.ini` file, you have the option of either reconfiguring your existing server, or removing the server instance by deleting the `my.ini` file and stopping and removing the MySQL service.

To reconfigure an existing server, choose the Re-configure Instance option and click the Next button. Your existing `my.ini` file is renamed to `my`*`timestamp`*`.ini.bak`, where *`timestamp`* is the date and time at which the existing `my.ini` file was created. To remove the existing server instance, choose the Remove Instance option and click the Next button.

If you choose the Remove Instance option, you advance to a confirmation window. Click the Execute button. The MySQL Configuration Wizard stops and removes the MySQL service, and then deletes the `my.ini` file. The server installation and its `data` folder are not removed.

If you choose the Re-configure Instance option, you advance to the Configuration Type dialog where you can choose the type of installation that you wish to configure.

2.3.4.4 Choosing a Configuration Type

When you start the MySQL Configuration Wizard for a new MySQL installation, or choose the Re-configure Instance option for an existing installation, you advance to the Configuration Type dialog.

There are two configuration types available: Detailed Configuration and Standard Configuration. The Standard Configuration option is intended for new users who want to get started with MySQL quickly without having to make a lot of decisions about server configuration. The `Detailed Configuration` option is intended for advanced users who want more fine-grained control of server configuration.

If you are new to MySQL and need a server configured as a single-user developer machine, the Standard Configuration should suit your needs. Choosing the Standard Configuration option causes the MySQL Configuration Wizard to automatically set all configuration options with the exception of the `Service Options` and `Security Options`.

The `Standard Configuration` sets options that may be incompatible with systems where there are existing MySQL installations. If you have an existing MySQL installation on your system in addition to the installation you wish to configure, the `Detailed Configuration` option is recommended.

To complete the Standard Configuration, please refer to the sections on Service Options and Security Options in Section 2.3.4.11, “The Service Options Dialog”, and Section 2.3.4.12, “The Security Options Dialog”, respectively.

2.3.4.5 The Server Type Dialog

There are three different server types available to choose from. The server type that you choose affects the decisions that the MySQL Configuration Wizard makes with regard to memory, disk, and processor usage.

- Developer Machine: Choose this option for a typical desktop workstation where MySQL is intended only for personal use. It is assumed that many other desktop applications are running. The MySQL server is configured to use minimal system resources.
- Server Machine: Choose this option for a server machine where the MySQL server is running alongside other server applications such as FTP, email, and Web servers. The MySQL server is configured to use a medium portion of the system resources.
- Dedicated MySQL Server Machine: Choose this option for a server machine that is intended to run only the MySQL server. It is assumed that no other applications are running. The MySQL server is configured to use all available system resources.

Note

By selecting one of the preconfigured configurations, the values and settings of various options in your `my.cnf` or `my.ini` will be altered accordingly. The default values and options as described in the reference manual may therefore be different to the options and values that were created during the execution of the configuration wizard.

2.3.4.6 The Database Usage Dialog

The Database Usage dialog enables you to indicate the storage engines that you expect to use when creating MySQL tables. The option you choose determines whether the `InnoDB` storage engine is available and what percentage of the server resources are available to `InnoDB`.

- Multifunctional Database: This option enables both the `InnoDB` and `MyISAM` storage engines, and divides resources evenly between the two. This option is recommended for users who use both table handlers on a regular basis.
- Transactional Database Only: This option enables both the `InnoDB` and `MyISAM` storage engines but dedicates most server resources to the `InnoDB` storage engine. This option is recommended for users who use `InnoDB` almost exclusively and make only minimal use of `MyISAM`.
- Non-Transactional Database Only: This option disables the `InnoDB` storage engine completely, and dedicates all server resources to the `MyISAM` storage engine. This option is recommended for users who do not use `InnoDB`.

The Configuration Wizard uses a template to generate the server configuration file. The Database Usage dialog sets one of the following option strings:

```
Multifunctional Database:        MIXED
Transactional Database Only:     INNODB
Non-Transactional Database Only: MYISAM
```

When these options are processed through the default template (my-template.ini) the result is:

```
Multifunctional Database:
default-storage-engine=InnoDB
_myisam_pct=50

Transactional Database Only:
default-storage-engine=InnoDB
_myisam_pct=5

Non-Transactional Database Only:
default-storage-engine=MyISAM
_myisam_pct=100
skip-innodb
```

The `_myisam_pct` value is used to calculate the percentage of resources dedicated to `MyISAM`. The remaining resources are allocated to `InnoDB`.

2.3.4.7 The InnoDB Tablespace Dialog

Some users may want to locate the `InnoDB` tablespace files in a location other than the MySQL server data directory. Placing the tablespace files in a separate location can be desirable if your system has available a storage device availablehas with higher capacity or higher performance, such as a RAID storage system.

To change the default location for the `InnoDB` tablespace files, choose a new drive from the drop-down list of drive letters and choose a new path from the drop-down list of paths. To create a custom path, click the ... button.

If you are modifying the configuration of an existing server, you must click the Modify button before you change the path. In this situation you must move existing tablespace files to the new location manually before starting the server.

2.3.4.8 The Concurrent Connections Dialog

To prevent the server from running out of resources, it is important to limit the number of concurrent connections to the MySQL server that can be established. The Concurrent Connections dialog enables you to choose the expected usage of your server, and sets the limit for concurrent connections accordingly. It is also possible to manually set the concurrent connection limit.

- Decision Support (DSS)/OLAP: Choose this option if the server does not require a large number of concurrent connections. The maximum number of connections is set at 100, with an average of 20 concurrent connections assumed.
- Online Transaction Processing (OLTP): Choose this option if the server requires a large number of concurrent connections. The maximum number of connections is set at 500.
- Manual Setting: Choose this option to set the maximum number of concurrent connections to the server manually. Choose the number of concurrent connections from the drop-down box provided, or type the maximum number of connections into the drop-down box if the number you desire is not listed.

2.3.4.9 The Networking and Strict Mode Options Dialog

Use the Networking Options dialog to enable or disable TCP/IP networking and to configure the port number that is used to connect to the MySQL server.

TCP/IP networking is enabled by default. To disable TCP/IP networking, uncheck the box next to the Enable TCP/IP Networking option.

Port 3306 is used by default. To change the port used to access MySQL, choose a new port number from the drop-down box or type a new port number directly into the drop-down box. If the port number you choose is in use, you are prompted to confirm your choice of port number.

Set the Server SQL Mode to either enable or disable strict mode. Enabling strict mode (default) makes MySQL behave more like other database management systems. *If you run applications that rely on MySQL's old "forgiving" behavior, make sure to either adapt those applications or to disable strict mode.* For more information about strict mode, see Section 5.1.6, "Server SQL Modes".

2.3.4.10 The Character Set Dialog

The MySQL server supports multiple character sets and it is possible to set a default server character set that is applied to all tables, columns, and databases unless overridden. Use the Character Set dialog to change the default character set of the MySQL server.

- Standard Character Set: Choose this option if you want to use `latin1` as the default server character set. `latin1` is used for English and many Western European languages.
- Best Support For Multilingualism: Choose this option if you want to use `utf8` as the default server character set. This is a Unicode character set that can store characters from many different languages.
- Manually Selected Default Character Set / Collation: Choose this option if you want to pick the server's default character set manually. Choose the desired character set from the provided drop-down list.

2.3.4.11 The Service Options Dialog

On Windows NT-based platforms, the MySQL server can be installed as a Windows service. When installed this way, the MySQL server can be started automatically during system startup, and even restarted automatically by Windows in the event of a service failure.

The MySQL Configuration Wizard installs the MySQL server as a service by default, using the service name `MySQL`. If you do not wish to install the service, un-check the box next to the Install As Windows Service option. You can change the service name by picking a new service name from the drop-down box provided or by typing a new service name into the drop-down box.

To install the MySQL server as a service but not have it started automatically at startup, un-check the box next to the Launch the MySQL Server automatically option.

2.3.4.12 The Security Options Dialog

It is strongly recommended that you set a `root` password for your MySQL server, and the MySQL Configuration Wizard requires by default that you do so. If you do not wish to set a `root` password, uncheck the box next to the Modify Security Settings option.

To set the `root` password, enter the desired password into both the New root password and Confirm boxes. If you are reconfiguring an existing server, you need to enter the existing `root` password into the Current root password box.

To prevent `root` logins from across the network, check the box next to the Root may only connect from localhost option. This increases the security of your `root` account.

To create an anonymous user account, check the box next to the Create An Anonymous Account option. Creating an anonymous account can decrease server security and cause login and permission difficulties and is not recommended.

2.3.4.13 The Confirmation Dialog

The final dialog in the MySQL Configuration Wizard is the Confirmation Dialog. To start the configuration process, click the Execute button. To return to a previous dialog, click the Back button. To exit the MySQL Configuration Wizard without configuring the server, click the Cancel button.

After you click the Execute button, the MySQL Configuration Wizard performs a series of tasks and displays the progress onscreen as the tasks are performed.

The MySQL Configuration Wizard firsts determines various configuration file options based on your choices using a template prepared by MySQL developers and engineers. This template is named `my-template.ini` and is located in your server installation directory.

The MySQL Configuration Wizard then writes these options to a `my.ini` file. The final location of the `my.ini` file is displayed next to the **Write configuration file** task.

If you chose to create a service for the MySQL server, the MySQL Configuration Wizard creates and starts the service. If you are reconfiguring an existing service, the MySQL Configuration Wizard restarts the service to apply your configuration changes.

If you chose to set a `root` password, the MySQL Configuration Wizard connects to the server, sets your new `root` password and applies any other security settings you may have selected.

After the MySQL Configuration Wizard has completed its tasks, it displays a summary. Click the Finish button to exit the MySQL Configuration Wizard.

2.3.4.14 The Location of the my.ini File

In MySQL installations prior to version 4.1.5 it was customary to name the server configuration file `my.cnf` or `my.ini` and locate the file either at `c:\my.cnf` or `c:\Windows\my.ini`.

The new MySQL Configuration Wizard places the `my.ini` file in the installation directory of the MySQL server. This helps associate configuration files with particular server instances.

To ensure that the MySQL server knows where to look for the `my.ini` file, an argument similar to this is passed to the MySQL server as part of the service installation:

```
--defaults-file="C:\Program Files\MySQL\MySQL Server 4.1\my.ini"
```

Here, *`C:\Program Files\MySQL\MySQL Server 4.1`* is replaced with the installation path to the MySQL Server. The `--defaults-file` [235] option instructs the MySQL server to read the specified file for configuration options when it starts.

2.3.4.15 Editing the my.ini File

To modify the `my.ini` file, open it with a text editor and make any necessary changes. You can also modify the server configuration with the http://www.mysql.com/products/administrator/ utility.

MySQL clients and utilities such as the `mysql` and `mysqldump` command-line clients are not able to locate the `my.ini` file located in the server installation directory. To configure the client and utility applications, create a new `my.ini` file in the `C:\WINDOWS` or `C:\WINNT` directory (whichever is applicable to your Windows version).

2.3.5 Installing MySQL from a Noinstall Zip Archive

Users who are installing from the Noinstall package, or who are installing a version of MySQL prior to 4.1.5 can use the instructions in this section to manually install MySQL. If you are installing a version prior

to 4.1.5 with an install package that includes a Setup program, substitute running the Setup program for extracting the archive.

The process for installing MySQL from a Zip archive is as follows:

1. Extract the archive to the desired install directory
2. Create an option file
3. Choose a MySQL server type
4. Start the MySQL server
5. Secure the default user accounts

This process is described in the sections that follow.

2.3.6 Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to Section 2.3.14, “Upgrading MySQL on Windows”, before beginning the upgrade process.
2. If you are using a Windows NT-based operating system such as Windows NT, Windows 2000, Windows XP, or Windows Server 2003, make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally, the MySQL server has been installed at `C:\mysql`. The MySQL Installation Wizard installs MySQL under `C:\Program Files\MySQL`. If you do not install MySQL in `C:\mysql`, you must specify the path to the install directory during startup or in an option file. See Section 2.3.7, “Creating an Option File”.
4. Extract the install archive to the chosen installation location using your preferred Zip archive tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.3.7 Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations differ from the default locations (`C:\mysql` and `C:\mysql\data`).
- You need to tune the server settings. For example, to use the `InnoDB` transactional tables in MySQL 3.23, you must manually add some extra lines to the option file, as described in Section 13.2.3, “`InnoDB` Configuration”. (As of MySQL 4.0, `InnoDB` creates its data files and log files in the data directory by default. This means you need not configure `InnoDB` explicitly. You may still do so if you wish, and an option file is also useful in this case.)

When the MySQL server starts on Windows, it looks for option files in several locations, such as the Windows directory, `C:\`, and the MySQL installation directory (for the full list of locations, see Section 4.2.3.3, “Using Option Files”). The Windows directory typically is named something like `C:\WINDOWS` or `C:\WINNT`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
shell> echo %WINDIR%
```

MySQL looks for options in each location first in the `my.ini` file, and then in the `my.cnf` file. However, to avoid confusion, it is best if you use only one file. If your PC uses a boot loader where the `C:` drive is not the boot drive, your only option is to use the `my.ini` file. Whichever option file you use, it must be a plain text file.

You can also make use of the example option files included with your MySQL distribution; see Preconfigured Option Files.

An option file can be created and modified with any text editor, such as the `Notepad` program. For example, if MySQL is installed in `E:\mysql` and the data directory is `E:\mydata\data`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Note that Windows path names are specified in option files using forward slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

The rules for use of backslash in option file values are given in Section 4.2.3.3, “Using Option Files”.

On Windows, the MySQL installer places the data directory directly under the directory where you install MySQL. If you would like to use a data directory in a different location, you should copy the entire contents of the `data` directory to the new location. For example, by default, the installer places MySQL in `C:\mysql`, and the data directory in `C:\mysql\data`. If you want to use `E:\mydata` as the data directory, you must do two things:

- Move the data directory from `C:\mysql\data` to `E:\mydata`.
- Use a `--datadir` [385] option to specify the new data directory location each time you start the server.

2.3.8 Selecting a MySQL Server Type

Starting with MySQL 3.23.38, the Windows distribution includes both the normal and the MySQL-Max server binaries.

Up through the early releases of MySQL 4.1, the servers included in Windows distributions are named like this:

Binary	Description
`mysqld`	Compiled with full debugging and automatic memory allocation checking, and `InnoDB` and `BDB` tables.
`mysqld-opt`	Optimized binary. From version 4.0 on, `InnoDB` is enabled. Before 4.0, this server includes no transactional table support.

Binary	Description
mysqld-nt	Optimized binary for Windows NT, 2000, and XP with named-pipe support.
mysqld-max	Optimized binary with InnoDB and BDB support.
mysqld-max-nt	Like mysqld-max, but compiled with named-pipe support.

We have found that the server with the most generic name (mysqld) is the one that many users are likely to choose by default. However, that is also the server that results in the highest memory and CPU use due to the inclusion of full debugging support. The server named mysqld-opt is a better general-use server choice to make instead if you do not need debugging support and do not want the maximal feature set offered by the -max servers or named pipe support offered by the -nt servers.

To make it less likely that the debugging server would be chosen inadvertently, some name changes were made from MySQL 4.1.2 to 4.1.4: mysqld has been renamed to mysqld-debug and mysqld-opt has been renamed to mysqld. Thus, the server that includes debugging support indicates that in its name, and the server named mysqld is an efficient default choice. The other servers still have their same names. The resulting servers are named like this:

Binary	Description
mysqld-debug	Compiled with full debugging and automatic memory allocation checking, and InnoDB and BDB tables.
mysqld	Optimized binary with InnoDB support.
mysqld-nt	Optimized binary for Windows NT, 2000, and XP with support for named pipes.
mysqld-max	Optimized binary with support for InnoDB and BDB tables.
mysqld-max-nt	Like mysqld-max, but compiled with support for named pipes.

The name changes were not both instituted at the same time. If you have MySQL 4.1.2 or 4.1.3, it might be that you have a server named mysqld-debug but not one named mysqld. In this case, you should have a server mysqld-opt, which you should choose as your default server unless you need maximal features, named pipes, or debugging support.

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

As of MySQL 4.0, all Windows servers have support for symbolic linking of database directories. Before MySQL 4.0, only the debugging and Max server versions include this feature.

MySQL supports TCP/IP on all Windows platforms. MySQL servers on Windows support named pipes as indicated in the following list. However, the default is to use TCP/IP regardless of platform. (Named pipes are slower than TCP/IP in many Windows configurations.)

Use of named pipes is subject to these conditions:

- Starting from MySQL 3.23.50, named pipes are enabled only if you start the server with the --enable-named-pipe [386] option. It is necessary to use this option explicitly because some users have experienced problems shutting down the MySQL server when named pipes were used.
- Named-pipe connections are permitted only by the mysqld-nt or mysqld-max-nt servers, and only if the server is run on a version of Windows that supports named pipes (NT, 2000, XP, 2003).
- These servers can be run on Windows 98 or Me, but only if TCP/IP is installed; named-pipe connections cannot be used.
- These servers cannot be run on Windows 95.

Note

Most of the examples in this manual use `mysqld` as the server name. If you choose to use a different server, such as `mysqld-nt`, make the appropriate substitutions in the commands that are shown in the examples.

2.3.9 Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the `Noinstall` version, or if you wish to configure and test MySQL manually rather than with the GUI tools.

On Windows 95, 98, or Me, MySQL clients always connect to the server using TCP/IP. (This enables any machine on your network to connect to your MySQL server.) Because of this, you must make sure that TCP/IP support is installed on your machine before starting MySQL. You can find TCP/IP on your Windows CD-ROM.

Note that if you are using an old Windows 95 release (for example, OSR2), it is likely that you have an old Winsock package; MySQL requires Winsock 2. You can get the newest Winsock from http://www.microsoft.com/. Windows 98 has the new Winsock 2 library, so it is unnecessary to update the library.

On NT-based systems such as Windows NT, 2000, XP, or 2003, clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections. For MySQL to work with TCP/IP on Windows NT 4, you must install service pack 3 (or newer).

In MySQL versions 4.1 and higher, Windows servers also support shared-memory connections if the server is started with the `--shared-memory` [392] option. Clients can connect through shared memory by using the `--protocol=MEMORY` [226] option.

For information about which server binary to run, see Section 2.3.8, “Selecting a MySQL Server Type”.

The examples in these sections assume that MySQL is installed under the default location of `C:\mysql`. Adjust the path names shown in the examples if you have MySQL installed in a different location.

Testing is best done from a command prompt in a console window (a “DOS window”). This way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
shell> C:\mysql\bin\mysqld --console
```

For a server that includes `InnoDB` support, you should see the messages similar to those following as it starts (the path names and sizes may differ):

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
```

```
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
Version: '4.0.14-log'  socket: ''  port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` [385] option, the server writes diagnostic output to the error log in the data directory (`C:\mysql\data` by default). The error log is the file with the `.err` extension.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

2.3.10 Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the `mysqld` server from the command line, you should start a console window (a “DOS window”) and enter this command:

```
shell> "C:\Program Files\MySQL\MySQL Server 4.1\bin\mysqld"
```

The path to `mysqld` may vary depending on the install location of MySQL on your system.

On non-NT versions of Windows, this command starts `mysqld` in the background. That is, after the server starts, you should see another command prompt. If you start the server this way on Windows NT, 2000, XP, or 2003, the server runs in the foreground and no command prompt appears until the server exits. Because of this, you should open another console window to run client programs while the server is running.

You can stop the MySQL server by executing this command:

```
shell> "C:\Program Files\MySQL\MySQL Server 4.1\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

If `mysqld` does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the `C:\mysql\data` directory. It is the file with a suffix of `.err`. You can also try to start the server as `mysqld --console`; in this case, you may get some useful information on the screen that may help solve the problem.

The last option is to start `mysqld` with the `--standalone` [393] and `--debug` [385] options. In this case, `mysqld` writes a log file `C:\mysqld.trace` that should contain the reason why `mysqld` doesn't start. See Section 18.4, “Porting to Other Systems”.

Use `mysqld --verbose --help` to display all the options that `mysqld` supports. (Prior to MySQL 4.1, omit the `--verbose` [395] option.)

2.3.11 Starting MySQL as a Windows Service

On the NT family (Windows NT, 2000, XP, 2003), the recommended way to run MySQL is to install it as a Windows service. With the MySQL server installed as a service, Windows starts and stops it server automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using `NET` commands, or with the graphical `Services` utility. Generally, to install MySQL as a Windows service you should be logged in using an account that has administrator rights.

The `Services` utility (the Windows `Service Control Manager`) can be found in the Windows Control Panel (under Administrative Tools on Windows 2000, XP, Vista and Server 2003). To avoid conflicts, it is advisable to close the `Services` utility while performing server installation or removal operations from the command line.

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
shell> C:\mysql\bin\mysqladmin -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

Install the server as a service using this command:

```
shell> C:\mysql\bin\mysqld --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

Before MySQL 4.0.2, no command-line arguments can be given following the `--install` option. MySQL 4.0.2 and up offers limited support for additional arguments:

- You can specify a service name immediately following the `--install` option. The default service name is `MySQL`.
- As of MySQL 4.0.3, if a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=`*`file_name`* [235] to specify the name of an option file from which the server should read options when it starts.

  The use of a single option other than `--defaults-file` [235] is possible but discouraged. `--defaults-file` [235] is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the a service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the group that has the same name as the service, and reads options from the standard option files.

  As of MySQL 4.0.17, the server also reads options from the `[mysqld]` group from the standard option files. This enables you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the same name as a service for use by the server installed with that service name.
- If the service-installation command specifies a `--defaults-file` [235] option after the service name, the server reads options only from the `[mysqld]` group of the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
shell> C:\mysql\bin\mysqld --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (`MySQL`) is given after the `--install` option. If no `--defaults-file` [235] option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` [235] option is present, the server reads options from the `[mysqld]` option group, and only from the named file.

You can also specify options in Start parameters in the Windows `Services` utility before you start the MySQL service.

Note

Prior to MySQL 4.0.17, a server installed as a Windows service has problems starting if its path name or the service name contains spaces. For this reason, with older versions, avoid installing MySQL in a directory such as `C:\Program Files` or using a service name containing spaces.

Once a MySQL server has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the `Services` utility, or by using a `NET START MySQL` command. The `NET` command is not case sensitive.

When run as a service, `mysqld` has no access to a console window, so no messages can be seen there. If `mysqld` does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, `C:\mysql\data`). It is the file with a suffix of `.err`.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually by using the `Services` utility, the `NET STOP MySQL` command, or the `mysqladmin shutdown` command.

From MySQL 3.23.44 on, you have the choice of installing the server as a **Manual** service if you do not wish the service to be started automatically during the boot process. To do this, use the `--install-manual` option rather than the `--install` option:

```
shell> C:\mysql\bin\mysqld --install-manual
```

To remove a server that is installed as a service, first stop it if it is running by executing `NET STOP MySQL`. Then use the `--remove` option to remove it:

```
shell> C:\mysql\bin\mysqld --remove
```

For MySQL versions older than 3.23.49, one problem with automatic MySQL service shutdown is that Windows waited only for a few seconds for the shutdown to complete, and then killed the database server process if the time limit was exceeded. This had the potential to cause problems. (For example, the `InnoDB` storage engine would have to perform crash recovery at the next startup.) Starting from MySQL 3.23.49, Windows waits longer for the MySQL server shutdown to complete. If you notice this still is not enough for your installation, it is safest not to run the MySQL server as a service. Instead, start it from the command-line prompt, and stop it with `mysqladmin shutdown`.

This change to tell Windows to wait longer when stopping the MySQL server works for Windows 2000 and XP. It does not work for Windows NT, where Windows waits only 20 seconds for a service to shut down, and after that kills the service process. You can increase this default by opening the `Registry Editor` (`\winnt\system32\regedt32.exe`) and editing the value of `WaitToKillServiceTimeout` at `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control` in the Registry tree. Specify the new larger value in milliseconds. For example, the value `120000` tells Windows NT to wait up to 120 seconds.

If `mysqld` is not running as a service, you can start it from the command line. For instructions, see Section 2.3.10, “Starting MySQL from the Windows Command Line”.

Please see Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”, if you encounter difficulties during installation.

2.3.12 Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
shell> C:\mysql\bin\mysqlshow
shell> C:\mysql\bin\mysqlshow -u root mysql
shell> C:\mysql\bin\mysqladmin version status proc
shell> C:\mysql\bin\mysql test
```

If `mysqld` is slow to respond to TCP/IP connections from client programs on Windows 9x/Me, there is probably a problem with your DNS. In this case, start `mysqld` with the `--skip-name-resolve` [393] option and use only `localhost` and IP addresses in the `Host` column of the MySQL grant tables.

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the `--pipe` [226] option or by specifying `.` (period) as the host name. Use the `--socket` [227] option to specify the name of the pipe if you do not want to use the default pipe name. As of MySQL 4.1, you can use the `--protocol=PIPE` [226] option instead.

Note that if you have set a password for the `root` account, deleted the anonymous account, or created a new user account, then you must use the appropriate `-u` and `-p` options with the commands shown above to connect with the MySQL Server. See Section 4.2.2, “Connecting to the MySQL Server”.

There are two versions of the MySQL command-line tool on Windows:

Binary	Description
`mysql`	Compiled on native Windows, offering limited text editing capabilities.
`mysqlc`	Compiled with the Cygnus GNU compiler and libraries, which offers `readline` editing. `mysqlc` was intended for use primarily with Windows 9x/Me. It does not support the updated authentication protocol used beginning with MySQL 4.1, and is not supported in

Binary	Description
	MySQL 4.1 and above. Beginning with MySQL 4.1.8, it is no longer included in MySQL Windows distributions.

To use `mysqlc`, you must have a copy of the `cygwinb19.dll` library installed somewhere that `mysqlc` can find it. If your distribution does not have the `cygwinb19.dll` library in the `bin` directory under the base directory of your MySQL installation, look for it in the `lib` directory and copy it to your Windows system directory (`\Windows\system` or a similar place).

For more information about `mysqlshow`, see Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

2.3.13 Troubleshooting a MySQL Installation Under Windows

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. The purpose of this section is to help you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\mysql\data`. See Section 5.3.1, “The Error Log”.

Another source of information regarding possible errors is the console messages displayed when the MySQL service is starting. Use the `NET START MySQL` command from the command line after installing `mysqld` as a service to see any error messages regarding the starting of the MySQL server as a service. See Section 2.3.11, “Starting MySQL as a Windows Service”.

The following examples show other common error messages you may encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, you may see these messages:

```
System error 1067 has occurred.
Fatal error: Can't open privilege tables: Table 'mysql.host' does not exist
```

  These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\mysql` and `C:\mysql\data`, respectively).

  This situation may occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, there may be old and new configuration files that conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

  If you have installed MySQL to a directory other than `C:\mysql`, you need to ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. The `my.ini` file needs to be located in your Windows directory, typically `C:\WINDOWS` or `C:\WINNT`. You can determine its exact location from the value of the `WINDIR` environment variable by issuing the following command from the command prompt:

```
shell> echo %WINDIR%
```

  An option file can be created and modified with any text editor, such as the `Notepad` program. For example, if MySQL is installed in `E:\mysql` and the data directory is `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

Note that Windows path names are specified in option files using forward slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\mysql
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

The rules for use of backslash in option file values are given in Section 4.2.3.3, “Using Option Files”.

If you change the `datadir` value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See Section 2.3.7, “Creating an Option File”.

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Configuration Wizard, you may see this error:

```
Error: Cannot create Windows service for MySql. Error: 0
```

This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This enables the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command-line:

```
shell> sc delete mysql
[SC] DeleteService SUCCESS
```

If the `sc` utility is not available for your version of Windows, download the `delsrv` utility from http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp and use the `delsrv mysql` syntax.

2.3.14 Upgrading MySQL on Windows

This section lists some of the steps you should take when upgrading MySQL on Windows.

1. Review Section 2.11.1, “Upgrading MySQL”, for additional information on upgrading MySQL that is not specific to Windows.
2. You should always back up your current MySQL installation before performing an upgrade. See Section 6.2, “Database Backup Methods”.
3. Download the latest Windows distribution of MySQL from http://dev.mysql.com.

4. Before upgrading MySQL, you must stop the server. If the server is installed as a service, stop the service with the following command from the command prompt:

   ```
   shell> NET STOP MySQL
   ```

   If you are not running the MySQL server as a service, use the following command to stop it:

   ```
   shell> C:\mysql\bin\mysqladmin -u root shutdown
   ```

   Note

   If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

5. Before upgrading to MySQL 4.1.5 or higher from a previous version, or from a version of MySQL installed from a Zip archive to a version of MySQL installed with the MySQL Installation Wizard, you must first manually remove the previous installation and MySQL service (if the server is installed as a service).

   To remove the MySQL service, use the following command:

   ```
   shell> C:\mysql\bin\mysqld --remove
   ```

   If you do not remove the existing service, the MySQL Installation Wizard may fail to properly install the new MySQL service.

6. If you are using the MySQL Installation Wizard, start the wizard as described in Section 2.3.3, “Using the MySQL Installation Wizard”.

7. If you are installing MySQL from a Zip archive, extract the archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql5`. Overwriting the existing installation is recommended.

8. If you were running MySQL as a Windows service and you had to remove the service earlier in this procedure, reinstall the service. (See Section 2.3.11, “Starting MySQL as a Windows Service”.)

9. Restart the server. For example, use `NET START MySQL` if you run MySQL as a service, or invoke `mysqld` directly otherwise.

10. If you encounter errors, see Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”.

2.4 Installing MySQL from RPM Packages on Linux

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages. The RPMs that we provide to the community should work on all versions of Linux that support RPM packages and use `glibc` 2.3. We also provide RPMs with binaries that are statically linked to a patched version of `glibc` 2.2, but only for the x86 (32-bit) architecture. To obtain RPM packages, see Section 2.1.3, “How to Get MySQL”.

For non-RPM Linux distributions, you can install MySQL using a `.tar.gz` package. See Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”.

We do provide some platform-specific RPMs; the difference between a platform-specific RPM and a generic RPM is that a platform-specific RPM is built on the targeted platform and is linked dynamically whereas a generic RPM is linked statically with LinuxThreads.

Note

RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by us in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

If you have problems with an RPM file (for example, if you receive the error `Sorry, the host 'xxxx' could not be looked up`), see Section 2.12.1.2, “Linux Binary Distribution Notes”.

In most cases, you need to install only the `MySQL-server` and `MySQL-client` packages to get a functional MySQL installation. The other packages are not required for a standard installation. If you want to run a MySQL-Max server that has additional capabilities, you should also install the `MySQL-Max` RPM. However, you should do so only *after* installing the `MySQL-server` RPM. See Section 5.2, “The `mysqld-max` Extended MySQL Server”.

For upgrades, if your installation was originally produced by installing multiple RPM packages, it is best to upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

If you get a dependency failure when trying to install the MySQL 4.0 packages (for example, `error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...`), you should also install the `MySQL-shared-compat` package, which includes both the shared libraries for backward compatibility (`libmysqlclient.so.12` for MySQL 4.0 and `libmysqlclient.so.10` for MySQL 3.23).

Some Linux distributions still ship with MySQL 3.23 and they usually link applications dynamically to save disk space. If these shared libraries are in a separate package (for example, `MySQL-shared`), it is sufficient to simply leave this package installed and just upgrade the MySQL server and client packages (which are statically linked and do not depend on the shared libraries). For distributions that include the shared libraries in the same package as the MySQL server (for example, Red Hat Linux), you could either install our 3.23 `MySQL-shared` RPM, or use the `MySQL-shared-compat` package instead. (Do not install both.)

The RPM packages shown in the following list are available. The names shown here use a suffix of `.glibc23.i386.rpm`, but particular packages can have different suffixes, as described later.

- `MySQL-server-`*`VERSION`*`.glibc23.i386.rpm`

  The MySQL server. You need this unless you only want to connect to a MySQL server running on another machine.

  Note: Server RPM files were called `MySQL-`*`VERSION`*`.i386.rpm` before MySQL 4.0.10. That is, they did not have `-server` in the name.

- `MySQL-Max-`*`VERSION`*`.i386.rpm`

  The MySQL-Max server. This server has additional capabilities that the one provided in the `MySQL-server` RPM does not. You must install the `MySQL-server` RPM first, because the `MySQL-Max` RPM depends on it.

- `MySQL-client-`*`VERSION`*`.glibc23.i386.rpm`

  The standard MySQL client programs. You probably always want to install this package.

- `MySQL-bench-`*`VERSION`*`.glibc23.i386.rpm`

  Tests and benchmarks. Requires Perl and the `DBI` and `DBD::mysql` modules.

- `MySQL-devel-VERSION.glibc23.i386.rpm`

  The libraries and include files that are needed if you want to compile other MySQL clients, such as the Perl modules.

- `MySQL-debuginfo-VERSION.glibc23.i386.rpm`

  This package contains debugging information. `debuginfo` RPMs are never needed to use MySQL software; this is true both for the server and for client programs. However, they contain additional information that might be needed by a debugger to analyze a crash.

- `MySQL-shared-VERSION.glibc23.i386.rpm`

  This package contains the shared libraries (`libmysqlclient.so*`) that certain languages and applications need to dynamically load and use MySQL. It contains single-threaded and thread-safe libraries. If you install this package, do not install the `MySQL-shared-compat` package.

- `MySQL-shared-compat-VERSION.glibc23.i386.rpm`

  This package includes the shared libraries for MySQL 3.23, 4.0, and so on, up to the current release. It contains single-threaded and thread-safe libraries. Install this package instead of `MySQL-shared` if you have applications installed that are dynamically linked against older versions of MySQL but you want to upgrade to the current version without breaking the library dependencies. This package has been available since MySQL 4.0.13.

- `MySQL-embedded-VERSION.glibc23.i386.rpm`

  The embedded MySQL server library (available as of MySQL 4.0).

- `MySQL-ndb-management-VERSION.glibc23.i386.rpm`, `MySQL-ndb-storage-VERSION.glibc23.i386.rpm`, `MySQL-ndb-tools-VERSION.glibc23.i386.rpm`, `MySQL-ndb-extra-VERSION.glibc23.i386.rpm`

  Packages that contain additional files for MySQL Cluster installations.

  Note

  The `MySQL-clustertools` RPM requires a working installation of perl and the `DBI` and `HTML::Template` packages. See Section 2.14, “Perl Installation Notes”, and Section 15.4.18, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”, for more information.

- `MySQL-VERSION.src.rpm`

  This contains the source code for all of the previous packages. It can also be used to rebuild the RPMs on other architectures (for example, Alpha or SPARC).

The suffix of RPM package names (following the *`VERSION`* value) has the following syntax:

```
[.PLATFORM].CPU.rpm
```

The *`PLATFORM`* and *`CPU`* values indicate the type of system for which the package is built. *`PLATFORM`*, if present, indicates the platform, and *`CPU`* indicates the processor type or family.

If the *`PLATFORM`* value is missing (for example, `MySQL-server-VERSION.i386.rpm`), the package is statically linked against a version of `glibc` 2.2 that has been patched to handle larger numbers of threads with larger stack sizes than the stock library. (The exception is that MySQL-Max RPMs are always dynamically linked.)

If *PLATFORM* is present, the package is dynamically linked against glibc 2.3 and the *PLATFORM* value indicates whether the package is platform independent or intended for a specific platform, as shown in the following table.

***PLATFORM* Value**	**Intended Use**
glibc23	Platform independent, should run on any Linux distribution that supports glibc 2.3
rhel3, rhel4	Red Hat Enterprise Linux 3 or 4
sles9, sles10	SuSE Linux Enterprise Server 9 or 10

The *CPU* value indicates the processor type or family for which the package is built.

***CPU* Value**	**Intended Processor Type or Family**
i386	x86 processor, 386 and up
i586	x86 processor, Pentium and up
x86_64	64-bit x86 processor
ia64	Itanium (IA-64) processor

To see all files in an RPM package (for example, a MySQL-server RPM), run a command like this:

```
shell> rpm -qpl MySQL-server-VERSION.glibc23.i386.rpm
```

To perform a standard minimal installation, install the server and client RPMs:

```
shell> rpm -i MySQL-server-VERSION.glibc23.i386.rpm
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

To install only the client programs, install just the client RPM:

```
shell> rpm -i MySQL-client-VERSION.glibc23.i386.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. If you would like to learn more about this feature, see Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

The server RPM places data under the /var/lib/mysql directory. The RPM also creates a login account for a user named mysql (if one does not exist) to use for running the MySQL server, and creates the appropriate entries in /etc/init.d/ to start the server automatically at boot time. (This means that if you have performed a previous installation and have made changes to its startup script, you may want to make a copy of the script so that you do not lose it when you install a newer RPM.) See Section 2.10.2.2, “Starting and Stopping MySQL Automatically”, for more information on how MySQL can be started automatically on system startup.

If the RPM files that you install include MySQL-server, the mysqld server should be up and running after installation. You should be able to start using MySQL.

If something goes wrong, you can find more information in the binary installation section. See Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

During RPM installation, a user named `mysql` and a group named `mysql` are created on the system. This is done using the `useradd`, `groupadd`, and `usermod` commands. Those commands require appropriate administrative privileges, which is ensured for locally managed users and groups (as listed in the `/etc/passwd` and `/etc/group` files) by the RPM installation process being run by `root`.

If you log in as the `mysql` user, you may find that MySQL displays “Invalid (old?) table or database name” errors that mention `.mysqlgui`, `lost+found`, `.mysqlgui`, `.bash_history`, `.fonts.cache-1`, `.lesshst`, `.mysql_history`, `.profile`, `.viminfo`, and similar files created by MySQL or operating system utilities. You can safely ignore these error messages or remove the files or directories that cause them if you do not need them.

For nonlocal user management (LDAP, NIS, and so forth), the administrative tools may require additional authentication (such as a password), and will fail if the installing user does not provide this authentication. Even if they fail, the RPM installation will not abort but succeed, and this is intentional. If they failed, some of the intended transfer of ownership may be missing, and it is recommended that the system administrator then manually ensures some appropriate user andgroup exists and manually transfers ownership following the actions in the RPM spec file.

2.5 Installing MySQL on Mac OS X

Beginning with MySQL 4.0.11, you can install MySQL on Mac OS X 10.3.x (“Panther”) or newer using a Mac OS X binary package in DMG format instead of the binary tarball distribution. Please note that older versions of Mac OS X (for example, 10.1.x or 10.2.x) are not supported by this package.

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

To obtain MySQL, see Section 2.1.3, “How to Get MySQL”.

Note

Before proceeding with the installation, be sure to shut down all running MySQL server instances by using either the MySQL Manager Application (on Mac OS X Server) or `mysqladmin shutdown` on the command line.

To install the MySQL DMG file, double-click the package icon. This launches the Mac OS X Package Installer, which guides you through the installation of MySQL.

Due to a bug in the Mac OS X package installer, you may see this error message in the destination disk selection dialog:

```
You cannot install this software on this disk. (null)
```

If this error occurs, simply click the Go Back button once to return to the previous screen. Then click Continue to advance to the destination disk selection again, and you should be able to choose the destination disk correctly. We have reported this bug to Apple and it is investigating this problem.

The Mac OS X DMG of MySQL installs itself into `/usr/local/mysql-`*`VERSION`* and also installs a symbolic link, `/usr/local/mysql`, that points to the new location. If a directory named `/usr/local/mysql` exists, it is renamed to `/usr/local/mysql.bak` first. In addition, the installer creates the grant tables in the `mysql` database by executing `mysql_install_db`.

The installation layout is similar to that of a `tar` file binary distribution; all MySQL binaries are located in the directory `/usr/local/mysql/bin`. The MySQL socket file is created as `/tmp/mysql.sock` by default. See Section 2.1.5, “Installation Layouts”.

MySQL installation requires a Mac OS X user account named `mysql`. A user account with this name should exist by default on Mac OS X 10.2 and up.

If you are running Mac OS X Server, a version of MySQL should already be installed. The following table shows the versions of MySQL that ship with Mac OS X Server versions.

Mac OS X Server Version	MySQL Version
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a

This manual section covers the installation of the official MySQL Mac OS X DMG only. Make sure to read Apple's help information about installing MySQL: Run the `Help View` application, select Mac OS X Server help, search for “MySQL”, and read the item entitled “Installing MySQL”.

For preinstalled versions of MySQL on Mac OS X Server, note especially that you should start `mysqld` with `safe_mysqld` instead of `mysqld_safe` if MySQL is older than version 4.0.

If you previously used Marc Liyanage's MySQL packages for Mac OS X from http://www.entropy.ch, you can simply follow the update instructions for packages using the binary installation layout as given on his pages.

If you are upgrading from Marc's 3.23.x versions or from the Mac OS X Server version of MySQL to the official MySQL DMG, you also need to convert the existing MySQL privilege tables to the current format, because some new security privileges have been added. See Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

If you want MySQL to start automatically during system startup, you also need to install the MySQL Startup Item. Starting with MySQL 4.0.15, it is part of the Mac OS X installation disk images as a separate installation package. Simply double-click the MySQLStartupItem.pkg icon and follow the instructions to install it. The Startup Item need be installed only once. There is no need to install it each time you upgrade the MySQL package later.

The Startup Item for MySQL is installed into `/Library/StartupItems/MySQLCOM`. (Before MySQL 4.1.2, the location was `/Library/StartupItems/MySQL`, but that collided with the MySQL Startup Item installed by Mac OS X Server.) Startup Item installation adds a variable `MYSQLCOM=-YES-` to the system configuration file `/etc/hostconfig`. If you want to disable the automatic startup of MySQL, simply change this variable to `MYSQLCOM=-NO-`.

On Mac OS X Server, the default MySQL installation uses the variable `MYSQL` in the `/etc/hostconfig` file. The MySQL Startup Item installer disables this variable by setting it to `MYSQL=-NO-`. This avoids boot time conflicts with the `MYSQLCOM` variable used by the MySQL Startup Item. However, it does not shut down a running MySQL server. You should do that yourself.

After the installation, you can start up MySQL by running the following commands in a terminal window. You must have administrator privileges to perform this task.

If you have installed the Startup Item, use this command:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
(Enter your password, if necessary)
(Press Control-D or enter "exit" to exit the shell)
```

For versions of MySQL older than 4.1.3, substitute `/Library/StartupItems/MySQLCOM/MySQLCOM` with `/Library/StartupItems/MySQL/MySQL` above.

If you do not use the Startup Item, enter the following command sequence:

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
(Enter your password, if necessary)
(Press Control-Z)
shell> bg
(Press Control-D or enter "exit" to exit the shell)
```

You should be able to connect to the MySQL server, for example, by running `/usr/local/mysql/bin/mysql`.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

For `tcsh`, use:

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see Section 4.2.1, “Invoking MySQL Programs”.

If you are upgrading an existing installation, note that installing a new MySQL DMG does not remove the directory of an older installation. Unfortunately, the Mac OS X Installer does not yet offer the functionality required to properly upgrade previously installed packages.

To use your existing databases with the new installation, you will need to copy the contents of the old data directory to the new data directory. Make sure that neither the old server nor the new one is running when you do this. After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.

2.6 Installing MySQL on Solaris

To obtain a binary MySQL distribution for Solaris in tarball or PKG format, http://dev.mysql.com/downloads/mysql/4.1.html.

If you install MySQL using a binary tarball distribution on Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

You can install MySQL on Solaris using a binary package in PKG format instead of the binary tarball distribution. Some basic PKG-handling commands follow:

- To add a package:

```
pkgadd -d package_name.pkg
```

- To remove a package:

```
pkgrm package_name
```

- To get a full list of installed packages:

```
pkginfo
```

- To get detailed information for a package:

```
pkginfo -l package_name
```

- To list the files belonging to a package:

```
pkgchk -v package_name
```

- To get packaging information for an arbitrary file:

```
pkgchk -l -p file_name
```

For additional information about installing MySQL on Solaris, see Section 2.12.3, “Solaris Notes”.

2.7 Installing MySQL on NetWare

Porting MySQL to NetWare was an effort spearheaded by Novell. Novell customers should be pleased to note that NetWare 6.5 ships with bundled MySQL binaries, complete with an automatic commercial use license for all servers running that version of NetWare.

MySQL for NetWare is compiled using a combination of `Metrowerks CodeWarrior for NetWare` and special cross-compilation versions of the GNU autotools.

The latest binary packages for NetWare can be obtained at http://dev.mysql.com/downloads/. See Section 2.1.3, “How to Get MySQL”.

To host MySQL, the NetWare server must meet these requirements:

- The latest Support Pack of NetWare 6.5 must be installed.
- The system must meet Novell's minimum requirements to run the respective version of NetWare.
- MySQL data and the program binaries must be installed on an NSS volume; traditional volumes are not supported.

To install MySQL for NetWare, use the following procedure:

1. If you are upgrading from a prior installation, stop the MySQL server. This is done from the server console, using the following command:

```
SERVER:  mysqladmin -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

2. Log on to the target server from a client machine with access to the location where you are installing MySQL.

3. Extract the binary package Zip file onto the server. Be sure to allow the paths in the Zip file to be used. It is safe to simply extract the file to `SYS:\`.

   If you are upgrading from a prior installation, you may need to copy the data directory (for example, `SYS:MYSQL\DATA`), as well as `my.cnf`, if you have customized it. You can then delete the old copy of MySQL.

4. You might want to rename the directory to something more consistent and easy to use. The examples in this manual use `SYS:MYSQL` to refer to the installation directory.

   Note that MySQL installation on NetWare does not detect if a version of MySQL is already installed outside the NetWare release. Therefore, if you have installed the latest MySQL version from the Web (for example, MySQL 4.1 or later) in `SYS:\MYSQL`, you must rename the folder before upgrading the NetWare server; otherwise, files in `SYS:\MySQL` are overwritten by the MySQL version present in NetWare Support Pack.

5. At the server console, add a search path for the directory containing the MySQL NLMs. For example:

   ```
   SERVER:  SEARCH ADD SYS:MYSQL\BIN
   ```

6. Initialize the data directory and the grant tables, if necessary, by executing `mysql_install_db` at the server console.

7. Start the MySQL server using `mysqld_safe` at the server console.

8. To finish the installation, you should also add the following commands to `autoexec.ncf`. For example, if your MySQL installation is in `SYS:MYSQL` and you want MySQL to start automatically, you could add these lines:

   ```
   #Starts the MySQL 4.0.x database server
   SEARCH ADD SYS:MYSQL\BIN
   MYSQLD_SAFE
   ```

   If you are running MySQL on NetWare 6.0, we strongly suggest that you use the `--skip-external-locking` [392] option on the command line:

   ```
   #Starts the MySQL 4.0.x database server
   SEARCH ADD SYS:MYSQL\BIN
   MYSQLD_SAFE --skip-external-locking
   ```

   It is also necessary to use `CHECK TABLE` and `REPAIR TABLE` instead of `myisamchk`, because `myisamchk` makes use of external locking. External locking is known to have problems on NetWare 6.0; the problem has been eliminated in NetWare 6.5. Note that the use of MySQL on Netware 6.0 is not officially supported.

   `mysqld_safe` on NetWare provides a screen presence. When you unload (shut down) the `mysqld_safe` NLM, the screen does not go away by default. Instead, it prompts for user input:

```
*<NLM has terminated; Press any key to close the screen>*
```

If you want NetWare to close the screen automatically instead, use the `--autoclose` [243] option to `mysqld_safe`. For example:

```
#Starts the MySQL 4.0.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --autoclose
```

The behavior of `mysqld_safe` on NetWare is described further in Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

9. When installing MySQL version 4.1.x or later, either for the first time or upgrading the 4.0.x version to 4.1.x or later, download and install the latest and appropriate Perl module and PHP extension for NetWare:
   - Perl: http://forge.novell.com/modules/xfcontent/downloads.php/perl/Modules/
   - PHP: http://forge.novell.com/modules/xfcontent/downloads.php/php/Modules/

If there was an existing installation of MySQL on the NetWare server, be sure to check for existing MySQL startup commands in `autoexec.ncf`, and edit or delete them as necessary.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

2.8 Installing MySQL from Generic Binaries on Other Unix-Like Systems

Oracle provides a set of binary distributions of MySQL. These include binary distributions in the form of compressed `tar` files (files with a `.tar.gz` extension) for a number of platforms, as well as binaries in platform-specific package formats for selected platforms.

This section covers the installation of MySQL from a compressed `tar` file binary distribution. For other platform-specific package formats, see the other platform-specific sections. For example, for Windows distributions, see Section 2.3, “Installing MySQL on Microsoft Windows”.

To obtain MySQL, see Section 2.1.3, “How to Get MySQL”.

MySQL compressed `tar` file binary distributions have names of the form `mysql-`*`VERSION`*`-`*`OS`*`.tar.gz`, where *`VERSION`* is a number (for example, `4.1.25`), and *`OS`* indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

To install MySQL from a compressed `tar` file binary distribution, your system must have GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.

If you run into problems and need to file a bug report, please use the instructions in Section 1.8, “How to Report Bugs or Problems”.

To install and use a MySQL binary distribution, the basic command sequence looks like this:

```
shell> groupadd mysql
shell> useradd -r -g mysql mysql
shell> cd /usr/local
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
shell> cp /usr/local/mysql/support-files/my-small/etc/my.cnf # Optional
shell> bin/mysqld_safe --user=mysql &
shell> cp /usr/local/mysql/support-files/mysql.server /etc/init.d/mysql.server # Optional
```

For versions of MySQL older than 4.0, substitute `bin/safe_mysqld` for `bin/mysqld_safe`.

A more detailed version of the preceding description for installing a binary distribution follows.

Note

This procedure assumes that you have `root` (administrator) access to your system. Alternatively, you can prefix each command using the `sudo` (Linux) or `pfexec` (OpenSolaris) command.

The procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.10, “Postinstallation Setup and Testing”.

Create a `mysql` User and Group

If your system does not already have a user and group for `mysqld` to run as, you may need to create one. The following commands add the `mysql` group and the `mysql` user. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

```
shell> groupadd mysql
shell> useradd -r -g mysql mysql
```

Note

Because the user is required only for ownership purposes, not login purposes, the `useradd` command uses the `-r` option to create a user that does not have login permissions to your server. Omit this option to permit logins for the user (or if your `useradd` does not support the option).

You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the preceding commands and in the following steps.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it. The example here unpacks the distribution under `/usr/local`. The instructions, therefore, assume that you have

permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as `root`.

```
shell> cd /usr/local
```

Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”. For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

Unpack the distribution, which creates the installation directory. Then create a symbolic link to that directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `tar` command creates a directory named `mysql-VERSION-OS`. The `ln` command makes a symbolic link to that directory. This enables you to refer more easily to the installation directory as `/usr/local/mysql`.

If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it. Replace the preceding `tar` command with the following alternative command to uncompress and extract the distribution:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
```

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For instructions, see Section 2.10, “Postinstallation Setup and Testing”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

2.9 Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see Section 2.1.1, “Operating Systems On Which MySQL Is Known To Run”.

Before you proceed with an installation from source, check whether we produce a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”.

To obtain a source distribution for MySQL, see Section 2.1.3, “How to Get MySQL”. MySQL source distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-VERSION.tar.gz`, `mysql-VERSION.zip`, or `mysql-VERSION.rpm`, where `VERSION` is a number like `4.1.25`.

To perform a MySQL installation using the source code:

- To build MySQL from source on Unix-like systems, including Linux, commercial Unix, BSD, Mac OS X and others using a `.tar.gz` or RPM-based source code distribution, see Section 2.9.1, “Installing MySQL from a Standard Source Distribution”.
- To build MySQL from source on Windows (Windows XP or newer required), see Section 2.9.7, “Installing MySQL from Source on Windows”.
- For information on building from one of our development trees, see Section 2.9.2, “Installing MySQL from a Development Source Tree”.
- For information on using the `configure` command to specify the source build parameters, including links to platform specific parameters that you might need, see Section 2.9.3, “MySQL Source-Configuration Options”.

To install MySQL from source, your system must have the following tools:

- GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it (if you use a `.tar.gz` distribution), or `WinZip` or another tool that can read `.zip` files (if you use a `.zip` distribution).

  GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.
- A working ANSI C++ compiler. GCC 3.2 or later, Sun Studio 10 or later, Visual Studio 2005 or later, and many current vendor-supplied compilers are known to work.
- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or newer. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- `libtool` 1.5, available from http://www.gnu.org/software/libtool/. 1.5.24 or later is recommended.

If you are using a version of `gcc` recent enough to understand the `-fno-exceptions` option, it is *very important* that you use this option. Otherwise, you may compile a binary that crashes randomly. Also use `-felide-constructors` and `-fno-rtti` as well. When in doubt, do the following:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
       -fno-exceptions -fno-rtti" ./configure \
       --prefix=/usr/local/mysql --enable-assembler \
       --with-mysqld-ldflags=-all-static
```

On most systems, this gives you a fast and stable binary.

If you run into problems and need to file a bug report, please use the instructions in Section 1.8, “How to Report Bugs or Problems”.

2.9.1 Installing MySQL from a Standard Source Distribution

To install MySQL from source, first configure, build, and install from a source package. Then follow the same postinstallation setup sequence as for a binary installation.

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have `rpmbuild`, use `rpm` instead.

```
```

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in Section 2.4, “Installing MySQL from RPM Packages on Linux”.

The sequence for installation from a compressed `tar` file source distribution is similar to the process for installing from a generic binary distribution that is detailed in Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”. For a MySQL `.tar.gz` source distribution, the basic installation command sequence looks like this:

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -g mysql mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> bin/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql var
shell> bin/mysqld_safe --user=mysql &
```

For versions of MySQL older than 4.0, substitute `bin/safe_mysqld` for `bin/mysqld_safe` in the final command.

Note

This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.10, “Postinstallation Setup and Testing”, for postinstallation setup and testing.

A more detailed version of the preceding description for installing MySQL from a source distribution follows:

1. Add a login user and group for `mysqld` to run as:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   These commands add the `mysql` group and the `mysql` user. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

   You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Perform the following steps as the `mysql` user, except as noted.

3. Pick the directory under which you want to unpack the distribution and change location into it.

4. Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”.

5. Unpack the distribution into the current directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

   ```
   shell> tar zxvf /path/to/mysql-VERSION.tar.gz
   ```

   This command creates a directory named `mysql-VERSION`.

   If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

   ```
   shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
   ```

6. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd mysql-VERSION
   ```

   Note that currently you must configure and build MySQL from this top-level directory. You cannot build it in a different directory.

7. Configure the release and compile everything:

   ```
   shell> ./configure --prefix=/usr/local/mysql
   shell> make
   ```

   When you run `configure`, you might want to specify other options. For example, if you need to debug `mysqld` or a MySQL client, run `configure` with the `--with-debug` [98] option, and then recompile and link your clients with the new client library. See Section 18.4, “Porting to Other Systems”.

   Run `./configure --help` for a list of options. Section 2.9.3, “MySQL Source-Configuration Options”, discusses some of the more useful options.

   If `configure` fails and you are going to send mail to a MySQL mailing list to ask for assistance, please include any lines from `config.log` that you think can help solve the problem. Also include the last couple of lines of output from `configure`. To file a bug report, please use the instructions in Section 1.8, “How to Report Bugs or Problems”.

   If the compile fails, see Section 2.9.4, “Dealing with Problems Compiling MySQL”, for help.

8. Install the distribution:

   ```
   shell> make install
   ```

   You might need to run this command as `root`.

   If you want to set up an option file, use one of those present in the `support-files` directory as a template. For example:

   ```
   shell> cp support-files/my-medium.cnf /etc/my.cnf
   ```

   You might need to run this command as `root`.

   If you want to configure support for `InnoDB` tables, you should edit the `/etc/my.cnf` file, removing the `#` character before the option lines that start with `innodb_...`, and modify the option values to be what you want. See Section 4.2.3.3, “Using Option Files”, and Section 13.2.3, “`InnoDB` Configuration”.

9. Change location into the installation directory:

```
shell> cd /usr/local/mysql
```

10. If you ran the `make install` command as `root`, the installed files will be owned by `root`. Ensure that the installation is accessible to `mysql` by executing the following commands as `root` in the installation directory:

```
shell> chown -R mysql .
shell> chgrp -R mysql .
```

    The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

11. If you have not installed MySQL before, you must create the MySQL data directory and initialize the grant tables:

```
shell> bin/mysql_install_db --user=mysql
```

    If you run the command as `root`, include the `--user` option as shown. If you run the command while logged in as `mysql`, you can omit the `--user` option.

    The command should create the data directory and its contents with `mysql` as the owner.

    Note that for MySQL versions older than 3.22.10, `mysql_install_db` left the server running after creating the grant tables. This is no longer true; you need to restart the server after performing the remaining steps in this procedure.

12. Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory:

```
shell> chown -R root .
shell> chown -R mysql var
```

13. If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` [423] read only to the server.

14. If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the script itself, and in Section 2.10.2.2, “Starting and Stopping MySQL Automatically”.

15. You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD::mysql` Perl modules. See Section 4.6.15, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”. For Perl module installation instructions, see Section 2.14, “Perl Installation Notes”.

After everything has been installed, test the distribution. To start the MySQL server, use the following command:

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

For versions of MySQL older than 4.0, substitute `safe_mysqld` for `mysqld_safe` in the command.

If you run the command as `root`, you should use the `--user` option as shown. The option value is the name of the login account that you created in the first step to use for running the server. If you run the `mysqld_safe` command while logged in as that user, you can omit the `--user` option.

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the *`host_name`*`.err` file in the data directory).

More information about `mysqld_safe` is given in Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

To make it more convenient to invoke programs installed in `/usr/local/mysql/bin`, you can add that directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. See Section 4.2.4, “Setting Environment Variables”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

2.9.2 Installing MySQL from a Development Source Tree

This section discusses how to install MySQL from the latest development source code. Development trees have not necessarily received the same level of testing as standard release distributions, so this installation method is usually required only if you need the most recent code changes. Do not use a development tree for production systems. If your goal is simply to get MySQL up and running on your system, you should use a standard release distribution (either a binary or source distribution). See Section 2.1.3, “How to Get MySQL”.

To obtain the source tree, you must have Bazaar installed. The Bazaar VCS Web site has instructions for downloading and installing Bazaar on different platforms. Bazaar is supported on any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows, or Mac OS X host.

MySQL development projects are hosted on Launchpad. MySQL projects, including MySQL Server, MySQL Workbench, and others are available from the Oracle/MySQL Engineering page. For the repositories related only to MySQL Server, see the MySQL Server page.

To build under Unix/Linux, you must have the following tools installed:

- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or newer. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- `autoconf` 2.58 (or newer), available from http://www.gnu.org/software/autoconf/.
- `automake` 1.8.1, available from http://www.gnu.org/software/automake/.
- `libtool` 1.5, available from http://www.gnu.org/software/libtool/. 1.5.24 or later is recommended.
- `m4`, available from http://www.gnu.org/software/m4/.
- `bison`, available from http://www.gnu.org/software/bison/. You should use the latest version of `bison` where possible. Versions 1.75 and 2.1 are known to work. There have been reported problems with `bison` 1.875. If you experience problems, upgrade to a later, rather than earlier, version. Versions of `bison` older than 1.75 may report this error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

To build under Windows you must have Microsoft Visual C++ 2005 Express Edition, Visual Studio .Net 2003 (7.1), or Visual Studio 2005 (8.0) compiler system.

Once the necessary tools are installed, create a local branch of the MySQL development tree on your machine using this procedure:

1. To obtain a copy of the MySQL source code, you must create a new Bazaar branch. If you do not already have a Bazaar repository directory set up, you must initialize a new directory:

   ```
   shell> mkdir mysql-server
   shell> bzr init-repo --trees mysql-server
   ```

   This is a one-time operation.

2. Assuming that you have an initialized repository directory, you can branch from the public MySQL server repositories to create a local source tree. To create a branch of a specific version:

   ```
   shell> cd mysql-server
   shell> bzr branch lp:mysql-server/4.1 mysql-4.1
   ```

   This is a one-time operation per source tree. You can branch the source trees for several versions of MySQL under the `mysql-server` directory.

3. The initial download will take some time to complete, depending on the speed of your connection. Please be patient. Once you have downloaded the first tree, additional trees should take significantly less time to download.

4. When building from the Bazaar branch, you may want to create a copy of your active branch so that you can make configuration and other changes without affecting the original branch contents. You can achieve this by branching from the original branch:

   ```
   shell> bzr branch mysql-4.1 mysql-4.1-build
   ```

5. To obtain changes made after you have set up the branch initially, update it using the `pull` option periodically. Use this command in the top-level directory of the local copy:

   ```
   shell> bzr pull
   ```

   You can examine the changeset comments for the tree by using the `log` option to `bzr`:

   ```
   shell> bzr log
   ```

   You can also browse changesets, comments, and source code online at the Launchpad MySQL Server page.

   If you see diffs (changes) or code that you have a question about, do not hesitate to send email to the MySQL `internals` mailing list. See Section 1.7.1, “MySQL Mailing Lists”. If you think you have a better idea on how to do something, send an email message to the list with a patch.

After you have the local branch, you can build MySQL server from the source code. On Windows, the build process is different from Unix/Linux: see Section 2.9.7, “Installing MySQL from Source on Windows”.

On Unix/Linux, use the `autoconf` system to create the `configure` script so that you can configure the build environment before building. The following example shows the typical commands required to build MySQL from a source tree.

1. Change location to the top-level directory of the source tree; replace `mysql-4.1` with the appropriate directory name.

   ```
   shell> cd mysql-4.1
   ```

2. Prepare the source tree for configuration.

   You must separately configure the `BDB` and `InnoDB` storage engines. Run the following commands from the main source directory:

   ```
   shell> (cd bdb/dist; sh s_all)
   shell> (cd innobase; autoreconf --force --install)
   ```

   You can omit the previous commands if you do not require `BDB` or `InnoDB` support.

   Prepare the remainder of the source tree:

   ```
   shell> autoreconf --force --install
   ```

   As an alternative to the preceding `autoreconf` command, you can use `BUILD/autorun.sh`, which acts as a shortcut for the following sequence of commands:

   ```
   shell> aclocal; autoheader
   shell> libtoolize --automake --force
   shell> automake --force --add-missing; autoconf
   shell> (cd bdb/dist; sh s_all)
   shell> (cd innobase; aclocal; autoheader; autoconf; automake)
   ```

   If you get some strange errors during this stage, verify that you have the correct version of `libtool` installed.

3. Configure the source tree and compile MySQL:

   ```
   shell> ./configure  # Add your favorite options here
   shell> make
   ```

   For a description of some `configure` options, see Section 2.9.3, “MySQL Source-Configuration Options”.

   A collection of configuration scripts is located in the `BUILD/` subdirectory. For example, you may find it more convenient to use the `BUILD/compile-pentium-debug` script than the preceding set of shell commands. To compile on a different architecture, modify the script by removing flags that are Pentium-specific, or use another script that may be more appropriate. These scripts are provided on an “as-is” basis. They are not supported and their contents may change from release to release.

4. When the build is done, run `make install`. Be careful with this on a production machine; the installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run `./configure` with values for the `--prefix` [95], `--with-tcp-port` [96], and `--with-unix-socket-path` [96] options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see Section 5.7, “Running Multiple MySQL Servers on the Same Machine”.

5. Play hard with your new installation. For example, try to make new features crash. Start by running `make test`. See Section 18.1.2, “The MySQL Test Suite”.

6. If you have gotten to the `make` stage, but the distribution does not compile, please enter the problem into our bugs database using the instructions given in Section 1.8, “How to Report Bugs or Problems”. If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a `command not found` error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

2.9.3 MySQL Source-Configuration Options

The `configure` script provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the `configure` command line. For a full list of options supported by `configure`, run this command:

```
shell> ./configure --help
```

You can also affect `configure` using certain environment variables. See Section 2.13, “Environment Variables”.

Some of the `configure` options available are described here. For options that may be of use if you have difficulties building MySQL, see Section 2.9.4, “Dealing with Problems Compiling MySQL”.

Many options configure compile-time defaults that can be overridden at server startup. For example, the `--prefix` [95], `--with-tcp-port` [96], and `with-unix-socket-path` [96] options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the `--basedir` [384], `--port` [391], and `--socket` [394] options for `mysqld`.

- To compile just the MySQL client libraries and client programs and not the server, use the `--without-server` [95] option:

  ```
  shell> ./configure --without-server
  ```

  If you have no C++ compiler, some client programs such as `mysql` cannot be compiled because they require C++. In this case, you can remove the code in `configure` that tests for the C++ compiler and then run `./configure` with the `--without-server` [95] option. The compile step should still try to build all clients, but you can ignore any warnings about files such as `mysql.cc`. (If `make` stops, try `make -k` to tell it to continue with the rest of the build even if errors occur.)

- To build the embedded MySQL library (`libmysqld.a`), use the `--with-embedded-server` [95] option.

- To place your log files and database directories elsewhere than under `/usr/local/var`, use a `configure` command something like one of these:

  ```
  shell> ./configure --prefix=/usr/local/mysql
  shell> ./configure --prefix=/usr/local \
             --localstatedir=/usr/local/mysql/data
  ```

  The first command changes the installation prefix so that everything is installed under `/usr/local/mysql` rather than the default of `/usr/local`. The second command preserves the default installation prefix, but overrides the default location for database directories (normally `/usr/local/var`) and changes it to `/usr/local/mysql/data`.

You can also specify the installation directory and data directory locations at server startup time by using the `--basedir` [384] and `--datadir` [385] options. These can be given on the command line or in an MySQL option file, although it is more common to use an option file. See Section 4.2.3.3, "Using Option Files".

- The `--with-tcp-port` [96] option specifies the port number on which the server listens for TCP/IP connections. The default is port 3306. To listen on a different port, use a `configure` command like this:

```
shell> ./configure --with-tcp-port=3307
```

- On Unix, if you want the MySQL socket file location to be somewhere other than the default location (normally in the directory `/tmp` or `/var/run`), use a `configure` command like this:

```
shell> ./configure \
          --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

The socket file name must be an absolute path name. You can also change the location of `mysql.sock` at server startup by using a MySQL option file. See Section B.5.4.5, "How to Protect or Change the MySQL Unix Socket File".

- To compile statically linked programs (for example, to make a binary distribution, to get better performance, or to work around problems with some Red Hat Linux distributions), run `configure` like this:

```
shell> ./configure --with-client-ldflags=-all-static \
          --with-mysqld-ldflags=-all-static
```

- If you are using `gcc` and do not have `libg++` or `libstdc++` installed, you can tell `configure` to use `gcc` as your C++ compiler:

```
shell> CC=gcc CXX=gcc ./configure
```

When you use `gcc` as your C++ compiler, it does not attempt to link in `libg++` or `libstdc++`. This may be a good thing to do even if you have those libraries installed. Some versions of them have caused strange problems for MySQL users in the past.

The following list indicates some compilers and environment variable settings that are commonly used with each one.

- `gcc` 2.7.2:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

- `gcc` 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti"
```

In most cases, you can get a reasonably optimized MySQL binary by using the options from the preceding list and adding the following options to the `configure` line:

```
--prefix=/usr/local/mysql --enable-assembler \
```

```
--with-mysqld-ldflags=-all-static
```

The full `configure` line would, in other words, be something like the following for all recent `gcc` versions:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The binaries we provide on the MySQL Web site at http://dev.mysql.com/downloads/ are all compiled with full optimization and should work well for most users. See Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”.

- If the build fails and produces errors about your compiler or linker not being able to create the shared library `libmysqlclient.so.N` (where `N` is a version number), you can work around this problem by giving the `--disable-shared` option to `configure`. In this case, `configure` does not build a shared `libmysqlclient.so.N` library.

- By default, MySQL uses the `latin1` (cp1252 West European) character set. To change the default set, use the `--with-charset` [97] option:

```
shell> ./configure --with-charset=CHARSET
```

`CHARSET` may be one of `big5`, `cp1251`, `cp1257`, `czech`, `danish`, `dec8`, `dos`, `euc_kr`, `gb2312`, `gbk`, `german1`, `hebrew`, `hp8`, `hungarian`, `koi8_ru`, `koi8_ukr`, `latin1`, `latin2`, `sjis`, `swe7`, `tis620`, `ujis`, `usa7`, or `win1251ukr`. (Additional character sets might be available. Check the output from `./configure --help` for the current list.)

As of MySQL 4.1.1, the default collation may also be specified. MySQL uses the `latin1_swedish_ci` collation. To change this, use the `--with-collation` [97] option:

```
shell> ./configure --with-collation=COLLATION
```

To change both the character set and the collation, use both the `--with-charset` [97] and `--with-collation` [97] options. The collation must be a legal collation for the character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.)

Warning

Before MySQL 4.1, if you change character sets after having created any tables, you have to run `myisamchk -r -q --set-character-set=charset_name` on every `MyISAM` table. Your indexes may be sorted incorrectly otherwise. This can happen if you install MySQL, create some tables, and then reconfigure MySQL to use a different character set and reinstall it.

With the `configure` option `--with-extra-charsets=LIST` [97], you can define which additional character sets should be compiled into the server. *`LIST`* is one of the following:

- A list of character set names separated by spaces
- `complex` to include all character sets that can't be dynamically loaded
- `all` to include all character sets into the binaries

Clients that want to convert characters between the server and the client should use the `SET NAMES` statement. See Section 9.1.4, “Connection Character Sets and Collations”.

- To configure MySQL with debugging code, use the `--with-debug` [98] option:

```
shell> ./configure --with-debug
```

This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See Section 18.4, “Porting to Other Systems”.

- If your client programs are using threads, you must compile a thread-safe version of the MySQL client library with the `--enable-thread-safe-client` [98] configure option. This creates a `libmysqlclient_r` library with which you should link your threaded applications. See Section 17.6.3.2, “Writing C API Threaded Client Programs”.

- Some features require that the server be built with compression library support, such as the `COMPRESS()` [866] and `UNCOMPRESS()` [869] functions, and compression of the client/server protocol. The `--with-zlib-dir=no|bundled|`*`DIR`* [98] option provides control over compression library support. The value `no` explicitly disables compression support. `bundled` causes the `zlib` library bundled in the MySQL sources to be used. A *`DIR`* path name specifies the directory in which to find the compression library sources.

- It is possible to build MySQL with big table support using the `--with-big-tables` [98] option, beginning with the following MySQL versions:

  - **4.0 series**: 4.0.25
  - **4.1 series**: 4.1.11

  This option causes the variables that store table row counts to be declared as `unsigned long long` rather than `unsigned long`. This enables tables to hold up to approximately 1.844E+19 ($(2^{32})^2$) rows rather than 2^{32} (~4.295E+09) rows. Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable this feature.

- See Section 2.12, “Operating System-Specific Notes”, for options that pertain to particular operating systems.

- See Section 5.6.6.2, “Using SSL Connections”, for options that pertain to configuring MySQL to support secure (encrypted) connections.

2.9.4 Dealing with Problems Compiling MySQL

All MySQL programs compile cleanly for us with no warnings on Solaris or Linux using `gcc`. On other systems, warnings may occur due to differences in system include files. See Section 2.9.6, “MIT-pthreads Notes”, for warnings that may occur when using MIT-pthreads. For other problems, check the following list.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If `configure` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `config.cache`. When `configure` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.

- Each time you run `configure`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before re-running `configure`:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run `make distclean`.

The following list describes some of the problems that have been found to occur most often when compiling MySQL:

- If you get errors such as the ones shown here when compiling `sql_yacc.cc`, you probably have run out of memory or swap space:

  ```
  Internal compiler error: program cc1plus got fatal signal 11
  Out of virtual memory
  Virtual memory exhausted
  ```

  The problem is that `gcc` requires a huge amount of memory to compile `sql_yacc.cc` with inline functions. Try running `configure` with the `--with-low-memory` [99] option:

  ```
  shell> ./configure --with-low-memory
  ```

  This option causes `-fno-inline` to be added to the compile line if you are using `gcc` and `-O0` if you are using something else. You should try the `--with-low-memory` [99] option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with-low-memory` [99] option usually fixes it.

- By default, `configure` picks `c++` as the compiler name and GNU `c++` links with `-lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:

  ```
  configure: error: installation or configuration problem:
  C++ compiler cannot create executables.
  ```

  You might also observe problems during compilation related to `g++`, `libg++`, or `libstdc++`.

  One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++`, or `libstdc++`. Take a look at the `config.log` file. It should contain the exact reason why your C++ compiler did not work. To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

  ```
  shell> CXX="gcc -O3" ./configure
  ```

  This works because `gcc` compiles C++ source files as well as `g++` does, but does not link in `libg++` or `libstdc++` by default.

  Another way to fix these problems is to install `g++`, `libg++`, and `libstdc++`. However, do not use `libg++` or `libstdc++` with MySQL because this only increases the binary size of `mysqld` without

providing any benefits. Some versions of these libraries have also caused strange problems for MySQL users in the past.

Using `gcc` as the C++ compiler is also required if you want to compile MySQL with RAID functionality (see Section 12.1.5, “CREATE TABLE Syntax”, for more info on RAID table type) and you are using GNU `gcc` version 3 and above. If you get errors like those following during the linking stage when you configure MySQL to compile with the option `--with-raid`, try to use `gcc` as your C++ compiler by defining the `CXX` environment variable:

```
gcc -O3 -DDBUG_OFF -rdynamic -o isamchk isamchk.o sort.o  libnisam.a
../mysys/libmysys.a ../dbug/libdbug.a ../strings/libmystrings.a
 -lpthread -lz -lcrypt -lnsl -lm -lpthread
../mysys/libmysys.a(raid.o)(.text+0x79): In function
`my_raid_create':: undefined reference to `operator new(unsigned)'
../mysys/libmysys.a(raid.o)(.text+0xdd): In function
`my_raid_create':: undefined reference to `operator delete(void*)'
../mysys/libmysys.a(raid.o)(.text+0x129): In function
`my_raid_open':: undefined reference to `operator new(unsigned)'
../mysys/libmysys.a(raid.o)(.text+0x189): In function
`my_raid_open':: undefined reference to `operator delete(void*)'
../mysys/libmysys.a(raid.o)(.text+0x64b): In function
`my_raid_close':: undefined reference to `operator delete(void*)'
collect2: ld returned 1 exit status
```

- To define flags to be used by your C or C++ compilers, specify them using the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

- If you get errors such as those shown here when compiling `mysqld`, `configure` did not correctly detect the type of the last argument to `accept()`, `getsockname()`, or `getpeername()`:

```
cxx: Error: mysqld.cc, line 645: In this statement, the referenced
     type of the pointer value ''length'' is ''unsigned long'',
     which is not compatible with ''int''.
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

To fix this, edit the `config.h` file (which is generated by `configure`). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `XXX` to `size_t` or `int`, depending on your operating system. (You must do this each time you run `configure` because `configure` regenerates `config.h`.)

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

Or:

```
make: file `Makefile' line 18: Must be a separator (:
```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome make programs.

GNU make 3.75 is known to work.

- The sql_yacc.cc file is generated from sql_yacc.yy. Normally, the build process does not need to create sql_yacc.cc because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

  ```
  "sql_yacc.yy", line xxx fatal: default action causes potential...
  ```

  This is a sign that your version of yacc is deficient. You probably need to install bison (the GNU version of yacc) and use that instead.

- On Debian Linux 3.0, you need to install gawk instead of the default mawk if you want to compile MySQL 4.1 or higher with Berkeley DB support.

- If you get a compilation error on Linux (for example, SuSE Linux 8.1 or Red Hat Linux 7.3) similar to the following one, you probably do not have g++ installed:

  ```
  libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
  incompatible pointer type
  libmysql.c:1329: too few arguments to function `gethostbyname_r'
  libmysql.c:1329: warning: assignment makes pointer from integer
  without a cast
  make[2]: *** [libmysql.lo] Error 1
  ```

  By default, the configure script attempts to determine the correct number of arguments by using g++ (the GNU C++ compiler). This test yields incorrect results if g++ is not installed. There are two ways to work around this problem:

  - Make sure that the GNU C++ g++ is installed. On some Linux distributions, the required package is called gpp; on others, it is named gcc-c++.

  - Use gcc as your C++ compiler by setting the CXX environment variable to gcc:

    ```
    export CXX="gcc"
    ```

  You must run configure again after making either of those changes.

For information about acquiring or updating tools, see the system requirements in Section 2.9, “Installing MySQL from Source”.

2.9.5 Compiling and Linking an Optimized mysqld Server

Most of the following tests were performed on Linux with the MySQL benchmarks, but they should give some indication for other operating systems and workloads.

You obtain the fastest executables when you link with -static.

By using better compiler and compilation options, you can obtain a 10% to 30% speed increase in applications. This is particularly important if you compile the MySQL server yourself.

When we tested both the Cygnus CodeFusion and Fujitsu compilers, neither was sufficiently bug-free to enable MySQL to be compiled with optimizations enabled.

The standard MySQL binary distributions are compiled with support for all character sets. When you compile MySQL yourself, you should include support only for the character sets that you are going to use. This is controlled by the `--with-charset` [97] option to `configure`.

Here is a list of some measurements that we have made:

- If you link dynamically (without `-static`), the result is 13% slower on Linux. Note that you still can use a dynamically linked MySQL library for your client applications. It is the server that is most critical for performance.
- For a connection from a client to a server running on the same host, if you connect using TCP/IP rather than a Unix socket file, performance is 7.5% slower. (On Unix, if you connect to the host name `localhost`, MySQL uses a socket file by default.)
- For TCP/IP connections from a client to a server, connecting to a remote server on another host is 8% to 11% slower than connecting to a server on the same host, even for connections faster than 100Mb/s Ethernet.
- When running our benchmark tests using secure connections (all data encrypted with internal SSL support) performance was 55% slower than with unencrypted connections.
- If you compile with `--with-debug=full` [98], most queries are 20% slower. Some queries may take substantially longer; for example, the MySQL benchmarks run 35% slower. If you use `--with-debug` [98] (without `=full`), the speed decrease is only 15%. For a version of `mysqld` that has been compiled with `--with-debug=full` [98], you can disable memory checking at runtime by starting it with the `--skip-safemalloc` [394] option. The execution speed should then be close to that obtained when configuring with `--with-debug` [98].
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster than one compiled with `gcc` 3.2.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster in 32-bit mode than in 64-bit mode.
- Compiling with `gcc` 2.95.2 for UltraSPARC with the `-mcpu=v8 -Wa,-xarch=v8plusa` options gives 4% more performance.
- On Solaris 2.5.1, MIT-pthreads is 8% to 12% slower than Solaris native threads on a single processor. With greater loads or more CPUs, the difference should be larger.
- Compiling on Linux-x86 using `gcc` without frame pointers (`-fomit-frame-pointer` or `-fomit-frame-pointer -ffixed-ebp`) makes `mysqld` 1% to 4% faster.

2.9.6 MIT-pthreads Notes

This section describes some of the issues involved in using MIT-pthreads.

On Linux, you should *not* use MIT-pthreads. Use the installed LinuxThreads implementation instead. See Section 2.12.1, “Linux Notes”.

If your system does not provide native thread support, you should build MySQL using the MIT-pthreads package. This includes older FreeBSD systems, SunOS 4.x, Solaris 2.4 and earlier, and some others. See Section 2.1.1, “Operating Systems On Which MySQL Is Known To Run”.

Beginning with MySQL 4.0.2, MIT-pthreads is no longer part of the source distribution. If you require this package, you need to download it separately from http://dev.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz

After downloading, extract this source archive into the top level of the MySQL source directory. It creates a new subdirectory named `mit-pthreads`.

- On most systems, you can force MIT-pthreads to be used by running `configure` with the `--with-mit-threads` option:

```
shell> ./configure --with-mit-threads
```

  Building in a nonsource directory is not supported when using MIT-pthreads because we want to minimize our changes to this code.

- The checks that determine whether to use MIT-pthreads occur only during the part of the configuration process that deals with the server code. If you have configured the distribution using `--without-server` [95] to build only the client code, clients do not know whether MIT-pthreads is being used and use Unix socket file connections by default. Because Unix socket files do not work under MIT-pthreads on some platforms, this means you need to use `-h` or `--host` with a value other than `localhost` when you run client programs.

- When MySQL is compiled using MIT-pthreads, system locking is disabled by default for performance reasons. You can tell the server to use system locking with the `--external-locking` [387] option. This is needed only if you want to be able to run two MySQL servers against the same data files, but that is not recommended, anyway.

- Sometimes the pthread `bind()` command fails to bind to a socket without any error message (at least on Solaris). The result is that all connections to the server fail. For example:

```
shell> mysqladmin version
mysqladmin: connect to server at '' failed;
error: 'Can't connect to mysql server on localhost (146)'
```

  The solution to this problem is to kill the `mysqld` server and restart it. This has happened to us only when we forced the server to shut down and then restarted it immediately.

- With MIT-pthreads, the `sleep()` system call isn't interruptible with `SIGINT` (break). This is noticeable only when you run `mysqladmin --sleep`. You must wait for the `sleep()` call to terminate before the interrupt is served and the process stops.

- When linking, you might receive warning messages like these (at least on Solaris); they can be ignored:

```
ld: warning: symbol `_iob' has differing sizes:
    (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
    /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
ld: warning: symbol `__iob' has differing sizes:
    (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
file /usr/lib/libc.so value=0x140);
    /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
```

- Some other warnings also can be ignored:

```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- We have not gotten `readline` to work with MIT-pthreads. (This is not needed, but may be interesting for someone.)

2.9.7 Installing MySQL from Source on Windows

These instructions describe how to build binaries from source for MySQL 4.1 on Windows. Instructions are provided building binaries from a standard source distribution or from the Bazaar tree that contains the latest development source.

Note

The instructions here are strictly for users who want to test MySQL on Microsoft Windows from the latest source distribution or from the Bazaar tree. For production use, we do not advise using a MySQL server built by yourself from source. Normally, it is best to use precompiled binary distributions of MySQL that are built specifically for optimal performance on Windows by Oracle Corporation. Instructions for installing binary distributions are available in Section 2.3, “Installing MySQL on Microsoft Windows”.

To build MySQL on Windows from source, you must satisfy the following system, compiler, and resource requirements:

- Windows XP, Windows 2000, or newer version
- Visual Studio .Net 2003 (7.1) compiler system
- 3GB to 5GB of disk space

You also need a MySQL source distribution for Windows, which can be obtained two ways:

- Obtain a Windows source distribution packaged for the particular version of MySQL in which you are interested. These are available from http://dev.mysql.com/downloads/.
- Package a source distribution yourself from the latest Bazaar developer source tree. If you plan to do this, you must create the package on a Unix system and then transfer it to your Windows system. (Some of the configuration and build steps require tools that work only on Unix.) The Bazaar approach thus requires:
  - A system running Unix, or a Unix-like system such as Linux.
  - Bazaar installed on that system. See Section 2.9.2, “Installing MySQL from a Development Source Tree”, for instructions how to download and install Bazaar.

If you are using a Windows source distribution, you can go directly to Section 2.9.7.1, “Building MySQL from Source Using VC++”. To build from the Bazaar tree, proceed to Section 2.9.7.2, “Creating a Windows Source Package from the Latest Development Source”.

If you find something not working as expected, or you have suggestions about ways to improve the current build process on Windows, please send a message to the `win32` mailing list. See Section 1.7.1, “MySQL Mailing Lists”.

2.9.7.1 Building MySQL from Source Using VC++

Note

VC++ workspace files for MySQL 4.1 and above are compatible with Microsoft Visual Studio 7.1 and tested by us before each release.

Follow this procedure to build MySQL:

1. Create a work directory (for example, `C:\workdir`).
2. Unpack the source distribution in the aforementioned directory using `WinZip` or other Windows tool that can read `.zip` files.
3. Start Visual Studio .Net 2003 (7.1).
4. From the File menu, select Open Solution....
5. Open the `mysql.sln` solution you find in the work directory.
6. From the Build menu, select Configuration Manager....
7. In the Active Solution Configuration pop-up menu, select the configuration to use. You likely want to use one of nt (normal server, not for Windows 98/ME), Max nt (more engines and features, not for 98/ME) or Debug configuration.
8. From the Build menu, select Build Solution.
9. Debug versions of the programs and libraries are placed in the `client_debug` and `lib_debug` directories. Release versions of the programs and libraries are placed in the `client_release` and `lib_release` directories.
10. Test the server. The server built using the preceding instructions expects that the MySQL base directory and data directory are `C:\mysql` and `C:\mysql\data` by default. If you want to test your server using the source tree root directory and its data directory as the base directory and data directory, you need to tell the server their path names. You can either do this on the command line with the `--basedir` [384] and `--datadir` [385] options, or by placing appropriate options in an option file. (See Section 4.2.3.3, “Using Option Files”.) If you have an existing data directory elsewhere that you want to use, you can specify its path name instead.
11. Start your server from the `client_release` or `client_debug` directory, depending on which server you built or want to use. The general server startup instructions are in Section 2.3, “Installing MySQL on Microsoft Windows”. You must adapt the instructions appropriately if you want to use a different base directory or data directory.
12. When the server is running in standalone fashion or as a service based on your configuration, try to connect to it from the `mysql` interactive command-line utility that exists in your `client_release` or `client_debug` directory.

When you are satisfied that the programs you have built are working correctly, stop the server. Then install MySQL as follows:

1. Create the directories where you want to install MySQL. For example, to install into `C:\mysql`, use these commands:

```
shell> mkdir C:\mysql
shell> mkdir C:\mysql\bin
shell> mkdir C:\mysql\data
shell> mkdir C:\mysql\share
shell> mkdir C:\mysql\scripts
```

   If you want to compile other clients and link them to MySQL, you should also create several additional directories:

```
shell> mkdir C:\mysql\include
```

```
shell> mkdir C:\mysql\lib
shell> mkdir C:\mysql\lib\debug
shell> mkdir C:\mysql\lib\opt
```

If you want to benchmark MySQL, create this directory:

```
shell> mkdir C:\mysql\sql-bench
```

Benchmarking requires Perl support. See Section 2.14, “Perl Installation Notes”.

2. From the `workdir` directory, copy into the `C:\mysql` directory the following directories:

```
shell> cd \workdir
C:\workdir> copy client_release\*.exe C:\mysql\bin
C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
C:\workdir> xcopy scripts\*.* C:\mysql\scripts /E
C:\workdir> xcopy share\*.* C:\mysql\share /E
```

If you want to compile other clients and link them to MySQL, you should also copy several libraries and header files:

```
C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

If you want to benchmark MySQL, you should also do this:

```
C:\workdir> xcopy sql-bench\*.* C:\mysql\bench /E
```

After installation, set up and start the server in the same way as for binary Windows distributions. See Section 2.3, “Installing MySQL on Microsoft Windows”.

2.9.7.2 Creating a Windows Source Package from the Latest Development Source

To create a Windows source package from the current Bazaar source tree, use the instructions here. This procedure must be performed on a system running a Unix or Unix-like operating system because some of the configuration and build steps require tools that work only on Unix. For example, the following procedure is known to work well on Linux.

1. Copy the Bazaar source tree for MySQL 4.1. For instructions on how to do this, see Section 2.9.2, “Installing MySQL from a Development Source Tree”.

2. Configure and build the distribution so that you have a server binary to work with. One way to do this is to run the following command in the top-level directory of your source tree:

```
shell> ./BUILD/compile-pentium-max
```

3. After making sure that the build process completed successfully, run the following utility script from top-level directory of your source tree:

```
shell> ./scripts/make_win_src_distribution
```

This script creates a Windows source package to be used on your Windows system. You can supply different options to the script based on your needs. See Section 4.4.2, "`make_win_src_distribution` — Create Source Distribution for Windows", for a list of permissible options.

By default, `make_win_src_distribution` creates a Zip-format archive with the name `mysql-VERSION-win-src.zip`, where *`VERSION`* represents the version of your MySQL source tree.

4. Copy or upload the Windows source package that you have just created to your Windows machine. To compile it, use the instructions in Section 2.9.7.1, "Building MySQL from Source Using VC++".

2.10 Postinstallation Setup and Testing

After installing MySQL, there are some issues that you should address. For example, on Unix, you should initialize the data directory and create the MySQL grant tables. On all platforms, an important security concern is that the initial accounts in the grant tables have no passwords. You should assign passwords to prevent unauthorized access to the MySQL server. Optionally, for MySQL 4.1.3 and up, you can create time zone tables to enable recognition of named time zones.

The following sections include postinstallation procedures that are specific to Windows systems and to Unix systems. Another section, Section 2.10.2.3, "Starting and Troubleshooting the MySQL Server", applies to all platforms; it describes what to do if you have trouble getting the server to start. Section 2.10.3, "Securing the Initial MySQL Accounts", also applies to all platforms. You should follow its instructions to make sure that you have properly protected your MySQL accounts by assigning passwords to them.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in Section 5.5, "The MySQL Access Privilege System", and Section 5.6, "MySQL User Account Management".

2.10.1 Windows Postinstallation Procedures

On Windows, you need not create the data directory and the grant tables. MySQL Windows distributions include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory. It is unnecessary to run the `mysql_install_db` script that is used on Unix. Regarding passwords, if you installed MySQL using the Windows Installation Wizard, you may have already assigned passwords to the accounts. (See Section 2.3.3, "Using the MySQL Installation Wizard".) Otherwise, use the password-assignment procedure given in Section 2.10.3, "Securing the Initial MySQL Accounts".

Before setting up passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see Section 2.3.9, "Starting the Server for the First Time"), and then issue the following commands to verify that you can retrieve information from the server. You may need to specify directory different from `C:\mysql\bin` on the command line. If you used the Windows Installation Wizard, the default directory is `C:\Program Files\MySQL\MySQL Server 4.1`, and the `mysql` and `mysqlshow` client programs are in `C:\Program Files\MySQL\MySQL Server 4.1\bin`. See Section 2.3.3, "Using the MySQL Installation Wizard", for more information.

Use `mysqlshow` to see what databases exist:

```
C:\> C:\mysql\bin\mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
```

```
| test      |
+-----------+
```

The list of installed databases may vary, but will always include the minimum of `mysql` and `information_schema`. In most cases, the `test` database will also be installed automatically.

The preceding command (and commands for other MySQL programs such as `mysql`) may not work if the correct MySQL account does not exist. For example, the program may fail with an error, or you may not be able to view all databases. If you installed using the MSI packages and used the MySQL Server Instance Config Wizard, then the `root` user will have been created automatically with the password you supplied. In this case, you should use the `-u root` and `-p` options. (You will also need to use the `-u root` and `-p` options if you have already secured the initial MySQL accounts.) With `-p`, you will be prompted for the `root` password. For example:

```
C:\> C:\mysql\bin\mysqlshow -u root -p
Enter password: (enter root password here)
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+
```

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM db" mysql
+------+-------+------+
| host | db    | user |
+------+-------+------+
| %    | test% |      |
+------+-------+------+
```

For more information about `mysqlshow` and `mysql`, see Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”, and Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”.

If you are running a version of Windows that supports services, you can set up the MySQL server to run automatically when Windows starts. See Section 2.3.11, “Starting MySQL as a Windows Service”.

2.10.2 Unix Postinstallation Procedures

After installing MySQL on Unix, you must initialize the grant tables, start the server, and make sure that the server works satisfactorily. You may also wish to arrange for the server to be started and stopped automatically when your system starts and stops. You should also assign passwords to the accounts in the grant tables.

On Unix, the grant tables are set up by the `mysql_install_db` program. For some installation methods, this program is run for you automatically if an existing database cannot be found.

- If you install MySQL on Linux using RPM distributions, the server RPM runs `mysql_install_db`.
- If you install MySQL on Mac OS X using a DMG distribution, the installer runs `mysql_install_db`.

For other platforms and installation types, including generic binary and source installs, you will need to run `mysql_install_db` yourself.

The following procedure describes how to initialize the grant tables (if that has not previously been done) and start the server. It also suggests some commands that you can use to test whether the server is accessible and working properly. For information about starting and stopping the server automatically, see Section 2.10.2.2, “Starting and Stopping MySQL Automatically”.

After you complete the procedure and have the server running, you should assign passwords to the accounts created by `mysql_install_db` and perhaps tighten access to test databases. For instructions, see Section 2.10.3, “Securing the Initial MySQL Accounts”.

In the examples shown here, the server runs under the user ID of the `mysql` login account. This assumes that such an account exists. Either create the account if it does not exist, or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location into the top-level directory of your MySQL installation, represented here by *`BASEDIR`*:

   ```
   shell> cd BASEDIR
   ```

   `BASEDIR` is the installation directory for your MySQL instance. It is likely to be something like `/usr/local/mysql` or `/usr/local`. The following steps assume that you have changed location to this directory.

   You will find several files and subdirectories in the *`BASEDIR`* directory. The most important for installation purposes are the `bin` and `scripts` subdirectories:

   - The `bin` directory contains client programs and the server. You should add the full path name of this directory to your `PATH` environment variable so that your shell finds the MySQL programs properly. See Section 2.13, “Environment Variables”.

     For some distribution types, `mysqld` is in the `libexec` directory.

   - The `scripts` directory contains the `mysql_install_db` script used to initialize the `mysql` database containing the grant tables that store the server access permissions.

     For some distribution types, `mysql_install_db` is in the `bin` directory.

2. If necessary, ensure that the distribution contents are accessible to `mysql`. If you unpacked the distribution as `mysql`, no further action is required. If you unpacked the distribution as `root`, its contents will be owned by `root`. Change its ownership to `mysql` by executing the following commands as `root` in the installation directory:

```
shell> chown -R mysql .
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the `mysql` user. The second changes the group attribute to the `mysql` group.

3. If necessary, run the `mysql_install_db` program to set up the initial MySQL grant tables containing the privileges that determine how users are permitted to connect to the server. You will need to do this if you used a distribution type for which the installation procedure does not run the program for you.

   Typically, `mysql_install_db` needs to be run only the first time you install MySQL, so you can skip this step if you are upgrading an existing installation, However, `mysql_install_db` does not overwrite any existing privilege tables, so it should be safe to run in any circumstances.

   The exact location of `mysql_install_db` will depends on the layout for your given installation. To initialize the grant tables, use one of the following commands, depending on whether `mysql_install_db` is located in the `bin` or `scripts` directory:

   ```
   shell> bin/mysql_install_db --user=mysql
   shell> scripts/mysql_install_db --user=mysql
   ```

   It might be necessary to specify other options such as `--basedir` [254] or `--datadir` [254] if `mysql_install_db` does not identify the correct locations for the installation directory or data directory. For example:

   ```
   shell> bin/mysql_install_db --user=mysql \
            --basedir=/opt/mysql/mysql \
            --datadir=/opt/mysql/mysql/data
   ```

   The `mysql_install_db` script creates the server's data directory with `mysql` as the owner. Under the data directory, it creates directories for the `mysql` database that holds the grant tables and the `test` database that you can use to test MySQL. The script also creates privilege table entries for `root` and anonymous-user accounts. The accounts have no passwords initially. Section 2.10.3, “Securing the Initial MySQL Accounts”, describes the initial privileges. Briefly, these privileges permit the MySQL `root` user to do anything, and permit anybody to create or use databases with a name of `test` or starting with `test_`.

   It is important to make sure that the database directories and files are owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this, the `--user` [255] option should be used as shown if you run `mysql_install_db` as `root`. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` [255] option from the command.

   `mysql_install_db` creates several tables in the `mysql` database, including `user`, `db`, `host`, `tables_priv`, `columns_priv`, `func`, and others. See Section 5.5, “The MySQL Access Privilege System”, for a complete listing and description of these tables.

   If you do not want to have the `test` database, you can remove it after starting the server, using the instructions in Section 2.10.3, “Securing the Initial MySQL Accounts”.

   If you have trouble with `mysql_install_db` at this point, see Section 2.10.2.1, “Problems Running `mysql_install_db`”.

For MySQL versions older than 3.22.10, `mysql_install_db` left the server running after creating the grant tables. This is no longer true; you need to restart the server after performing the remaining steps in this procedure.

4. Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory. For some distribution types, the data directory might be named `var` rather than `data`; adjust the second command accordingly.

```
shell> chown -R root .
shell> chown -R mysql data
```

5. If the plugin directory (the directory named by the `plugin_dir` [423] system variable) is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` [423] read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely.

6. If you installed MySQL using a source distribution, you may want to optionally copy one of the provided configuration files from the `support-files` directory into your `/etc` directory. There are different sample configuration files for different use cases, server types, and CPU and RAM configurations. If you want to use one of these standard files, you should copy it to `/etc/my.cnf`, or `/etc/mysql/my.cnf` and edit and check the configuration before starting your MySQL server for the first time.

   If you do not copy one of the standard configuration files, the MySQL server will be started with the default settings.

   If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `mysql.server` script itself, and in Section 2.10.2.2, “Starting and Stopping MySQL Automatically”.

7. Start the MySQL server:

```
shell> bin/mysqld_safe --user=mysql &
```

   For versions of MySQL older than 4.0, substitute `bin/safe_mysqld` for `bin/mysqld_safe`.

   It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this, the `--user` option should be used as shown if you run `mysqld_safe` as `root`. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` option from the command.

   Further instructions for running MySQL as an unprivileged user are given in Section 5.4.6, “How to Run MySQL as a Normal User”.

   If the command fails immediately and prints `mysqld ended`, you can find some information in the *`host_name`*`.err` file in the data directory.

   If you neglected to create the grant tables by running `mysql_install_db` before proceeding to this step, the following message appears in the error log file when you start the server:

```
mysqld: Can't find file: 'host.frm'
```

This error also occurs if you run `mysql_install_db` as `root` without the `--user` [255] option. Remove the `data` directory and run `mysql_install_db` with the `--user` [255] option as described previously.

If you have other problems starting the server, see Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”. For more information about `mysqld_safe`, see Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

8. Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin  Ver 14.7 Distrib 4.1.19, for linux on i586
...

Server version          4.1.19-max
Protocol version        10
Connection              Localhost via Unix socket
TCP port                3306
UNIX socket             var/lib/mysql/mysql.sock
Uptime:                 5 days 19 hours 19 min 0 sec

Threads: 1  Questions: 163  Slow queries: 0
Opens: 11  Flush tables:1  Open tables: 0  Queries per second avg: 0.007
Threads: 1  Questions: 9  Slow queries: 0
```

To see what else you can do with `mysqladmin`, invoke it with the `--help` [280] option.

9. Verify that you can shut down the server:

```
shell> bin/mysqladmin -u root shutdown
```

10. Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

```
shell> bin/mysqld_safe --user=mysql --log &
```

If `mysqld_safe` fails, see Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”.

11. Run some simple tests to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```
shell> bin/mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+

shell> bin/mysqlshow mysql
```

```
Database: mysql
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+--------------+

shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
+------+--------+------+
| host | db     | user |
+------+--------+------+
| %    | test   |      |
| %    | test_% |      |
+------+--------+------+
```

12. There is a benchmark suite in the `sql-bench` directory (under the MySQL installation directory) that you can use to compare how MySQL performs on different platforms. The benchmark suite is written in Perl. It requires the Perl DBI module that provides a database-independent interface to the various databases, and some other additional Perl modules:

    ```
    DBI
    DBD::mysql
    Data::Dumper
    Data::ShowTable
    ```

    These modules can be obtained from CPAN (http://www.cpan.org/). See also Section 2.14.1, “Installing Perl on Unix”.

    The `sql-bench/Results` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

    ```
    shell> cd sql-bench
    shell> perl run-all-tests
    ```

    If you do not have the `sql-bench` directory, you probably installed MySQL using RPM files other than the source RPM. (The source RPM includes the `sql-bench` benchmark directory.) In this case, you must first install the benchmark suite before you can use it. Beginning with MySQL 3.22, there are separate benchmark RPM files named `mysql-bench-`*`VERSION`*`.i386.rpm` that contain benchmark code and data.

    If you have a source distribution, there are also tests in its `tests` subdirectory that you can run. For example, to run `auto_increment.tst`, execute this command from the top-level directory of your source distribution:

    ```
    shell> mysql -vvf test < ./tests/auto_increment.tst
    ```

    The expected result of the test can be found in the `./tests/auto_increment.res` file.

13. At this point, you should have the server running. However, none of the initial MySQL accounts have a password, and the server permits permissive access to test databases. To tighten security, follow the instructions in Section 2.10.3, “Securing the Initial MySQL Accounts”.

As of MySQL 4.1.3, the installation procedure creates time zone tables in the `mysql` database but does not populate them. To do so, use the instructions in Section 9.7, “MySQL Server Time Zone Support”.

You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD::mysql` Perl modules. See Section 4.6.15, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”. For Perl module installation instructions, see Section 2.14, “Perl Installation Notes”.

If you would like to use `mysqlaccess` and have the MySQL distribution in some nonstandard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `bin/mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error will occur when you run `mysqlaccess`.

Note

On Windows, you can also perform the process described in this section using the Configuration Wizard (see Section 2.3.4.12, “The Security Options Dialog”). On other platforms, the MySQL distribution includes `mysql_secure_installation`, a command-line utility that automates much of the process of securing a MySQL installation.

2.10.2.1 Problems Running `mysql_install_db`

The purpose of the `mysql_install_db` script is to generate new MySQL privilege tables. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

If you want to re-create your privilege tables, first stop the `mysqld` server if it is running. Then rename the `mysql` directory under the data directory to save it, and then run `mysql_install_db`. Suppose that your current directory is the MySQL installation directory and that `mysql_install_db` is located in the `bin` directory and the data directory is named `data`. To rename the `mysql` database and re-run `mysql_install_db`, use these commands.

```
shell> mv data/mysql data/mysql.old
shell> bin/mysql_install_db --user=mysql
```

When you run `mysql_install_db`, you might encounter the following problems:

- **`mysql_install_db` fails to install the grant tables**

  You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

  ```
  Starting mysqld daemon with databases from XXXXXX
  mysqld ended
  ```

  In this case, you should examine the error log file very carefully. The log should be located in the directory `XXXXXX` named by the error message and should indicate why `mysqld` did not start. If you do not understand what happened, include the log when you post a bug report. See Section 1.8, “How to Report Bugs or Problems”.

- **There is a `mysqld` process running**

  This indicates that the server is running, in which case the grant tables have probably been created already. If so, there is no need to run `mysql_install_db` at all because it needs to be run only once (when you install MySQL the first time).

- **Installing a second `mysqld` server does not work when one server is running**

  This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

  ```
  Can't start server: Bind on TCP/IP port:
  Address already in use
  Can't start server: Bind on unix socket...
  ```

  For instructions on setting up multiple servers, see Section 5.7, “Running Multiple MySQL Servers on the Same Machine”.

- **You do not have write access to the `/tmp` directory**

  If you do not have write access to create temporary files or a Unix socket file in the default location (the `/tmp` directory), an error occurs when you run `mysql_install_db` or the `mysqld` server.

  You can specify different locations for the temporary directory and Unix socket file by executing these commands prior to starting `mysql_install_db` or `mysqld`, where *`some_tmp_dir`* is the full path name to some directory for which you have write permission:

  ```
  shell> TMPDIR=/some_tmp_dir/
  shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
  shell> export TMPDIR MYSQL_UNIX_PORT
  ```

  Then you should be able to run `mysql_install_db` and start the server with these commands:

  ```
  shell> bin/mysql_install_db --user=mysql
  shell> bin/mysqld_safe --user=mysql &
  ```

  If `mysql_install_db` is located in the `scripts` directory, modify the first command to `scripts/mysql_install_db`.

  See Section B.5.4.5, “How to Protect or Change the MySQL Unix Socket File”, and Section 2.13, “Environment Variables”.

There are some alternatives to running the `mysql_install_db` script provided in the MySQL distribution:

- If you want the initial privileges to be different from the standard defaults, you can modify `mysql_install_db` before you run it. However, it is preferable to use `GRANT` and `REVOKE` to change the privileges *after* the grant tables have been set up. In other words, you can run `mysql_install_db`, and then use `mysql -u root mysql` to connect to the server as the MySQL `root` user so that you can issue the necessary `GRANT` and `REVOKE` statements.

  If you want to install MySQL on several machines with the same privileges, you can put the `GRANT` and `REVOKE` statements in a file and execute the file as a script using `mysql` after running `mysql_install_db`. For example:

  ```
  shell> bin/mysql_install_db --user=mysql
  shell> bin/mysql -u root < your_script_file
  ```

  By doing this, you can avoid having to issue the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you are just learning how to use `GRANT` and `REVOKE` and have made so many modifications after running `mysql_install_db` that you want to wipe out the tables and start over.

  To re-create the grant tables, remove all the `.frm`, `.MYI`, and `.MYD` files in the `mysql` database directory. Then run the `mysql_install_db` script again.

  Note

  For MySQL versions older than 3.22.10, you should not delete the `.frm` files. If you accidentally do this, you should copy them back into the `mysql` directory from your MySQL distribution before running `mysql_install_db`.

- You can start `mysqld` manually using the `--skip-grant-tables` [392] option and add the privilege information yourself using `mysql`:

```
shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
shell> bin/mysql mysql
```

  From `mysql`, manually execute the SQL commands contained in `mysql_install_db`. Make sure that you run `mysqladmin flush-privileges` or `mysqladmin reload` afterward to tell the server to reload the grant tables.

  Note that by not using `mysql_install_db`, you not only have to populate the grant tables manually, you also have to create them first.

2.10.2.2 Starting and Stopping MySQL Automatically

Generally, you start the `mysqld` server in one of these ways:

- Invoke `mysqld` directly. This works on any platform.
- Run the MySQL server as a Windows service. This can be done on versions of Windows that support services (such as NT, 2000, XP, and 2003). The service can be set to start the server automatically when Windows starts, or as a manual service that you start on request. For instructions, see Section 2.3.11, “Starting MySQL as a Windows Service”.
- Invoke `mysqld_safe`, which tries to determine the proper options for `mysqld` and then runs it with those options. This script is used on Unix and Unix-like systems. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.
- Invoke `mysql.server`. This script is used primarily at system startup and shutdown on systems that use System V-style run directories (that is, `/etc/init.d` and run-level specific directories), where it usually is installed under the name `mysql`. The `mysql.server` script starts the server by invoking `mysqld_safe`. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.
- On Mac OS X, install a separate MySQL Startup Item package to enable the automatic startup of MySQL on system startup. The Startup Item starts the server by invoking `mysql.server`. See Section 2.5, “Installing MySQL on Mac OS X”, for details.

The `mysqld_safe` and `mysql.server` scripts and the Mac OS X Startup Item can be used to start the server manually, or automatically at system startup time. `mysql.server` and the Startup Item also can be used to stop the server.

To start or stop the server manually using the `mysql.server` script, invoke it with `start` or `stop` arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

Before `mysql.server` starts the server, it changes location to the MySQL installation directory, and then invokes `mysqld_safe`. If you want the server to run as some specific user, add an appropriate `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file, as shown later in this section. (It is possible that you will need to edit `mysql.server` if you've installed a binary distribution of MySQL in a nonstandard location. Modify it to change location into the proper directory before it runs `mysqld_safe`. If you do this, your modified version of `mysql.server` may be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.)

`mysql.server stop` stops the server by sending a signal to it. You can also stop the server manually by executing `mysqladmin shutdown`.

To start and stop MySQL automatically on your server, you need to add start and stop commands to the appropriate places in your `/etc/rc*` files.

If you use the Linux server RPM package (`MySQL-server-VERSION.rpm`), the `mysql.server` script is installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See Section 2.4, “Installing MySQL from RPM Packages on Linux”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as `mysqld`.

If you install MySQL from a source distribution or using a binary distribution format that does not install `mysql.server` automatically, you can install it manually. The script can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree.

To install `mysql.server` manually, copy it to the `/etc/init.d` directory with the name `mysql`, and then make it executable. Do this by changing location into the appropriate directory where `mysql.server` is located and executing these commands:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

Note

Older Red Hat systems use the `/etc/rc.d/init.d` directory rather than `/etc/init.d`. Adjust the preceding commands accordingly. Alternatively, first create `/etc/init.d` as a symbolic link that points to `/etc/rc.d/init.d`:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use `chkconfig`:

```
shell> chkconfig --add mysql
```

On some Linux systems, the following command also seems to be necessary to fully enable the `mysql` script:

```
shell> chkconfig --level 345 mysql on
```

On FreeBSD, startup scripts generally should go in `/usr/local/etc/rc.d/`. The `rc(8)` manual page states that scripts in this directory are executed only if their basename matches the `*.sh` shell file name pattern. Any other files or directories present within the directory are silently ignored. In other words, on

FreeBSD, you should install the `mysql.server` script as `/usr/local/etc/rc.d/mysql.server.sh` to enable automatic startup.

As an alternative to the preceding setup, some operating systems also use `/etc/rc.local` or `/etc/init.d/boot.local` to start additional services on startup. To start up MySQL using this method, you could append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

For other systems, consult your operating system documentation to see how to install startup scripts.

You can add options for `mysql.server` in a global `/etc/my.cnf` file. A typical `/etc/my.cnf` file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

The `mysql.server` script supports the following options: `basedir`, `datadir`, and `pid-file`. If specified, they *must* be placed in an option file, not on the command line. `mysql.server` supports only `start` and `stop` as command-line arguments.

The following table shows which option groups the server and each startup script read from option files.

Table 2.5 MySQL Startup scripts and supported server option groups

Script	Option Groups
`mysqld`	`[mysqld]`, `[server]`, `[mysqld-major_version]`
`mysqld_safe`	`[mysqld]`, `[server]`, `[mysqld_safe]`
`mysql.server`	`[mysqld]`, `[mysql.server]`, and as of MySQL 4.1.1, `[server]`

`[mysqld-major_version]` means that groups with names like `[mysqld-4.0]`, `[mysqld-4.1]`, and `[mysqld-5.0]` are read by servers having versions 4.0.x, 4.1.x, 5.0.x, and so forth. This feature was added in MySQL 4.0.14. It can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. However, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead when you begin using MySQL 4.0 or later.

For more information on MySQL configuration files and their structure and contents, see Section 4.2.3.3, “Using Option Files”.

2.10.2.3 Starting and Troubleshooting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server on Unix. If you are using Windows, see Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”.

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start.
- Specify any special options needed by the storage engines you are using.

- Make sure that the server knows where to find the data directory.
- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must be set such that the server can read and modify them.
- Verify that the network interfaces the server wants to use are available.

Some storage engines have options that control their behavior. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (`InnoDB`, `BDB`, `NDB`), be sure that you have them configured the way you want before starting the server:

- If you are using `InnoDB` tables, refer to the `InnoDB`-specific startup options. In MySQL 3.23, you must configure `InnoDB` explicitly or the server fails to start. From MySQL 4.0 on, `InnoDB` uses default values for its configuration options if you specify none. See Section 13.2.3, “`InnoDB` Configuration”.
- If you are using `BDB` (Berkeley DB) tables, see Section 13.5.3, “`BDB` Startup Options”.
- If you are using MySQL Cluster, see Section 15.3, “MySQL Cluster Configuration”.

Storage engines will use default option values if you specify none, but it is recommended that you review the available options and specify explicit values for those for which the defaults are not appropriate for your installation.

When the `mysqld` server starts, it changes location to the data directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

The data directory location is hardwired in when the server is compiled. This is where the server looks for the data directory by default. If the data directory is located somewhere else on your system, the server will not work properly. You can determine what the default path settings are by invoking `mysqld` with the `--verbose` [395] and `--help` [384] options. (Prior to MySQL 4.1, omit the `--verbose` [395] option.)

If the default locations do not match the MySQL installation layout on your system, you can override them by specifying options to `mysqld` or `mysqld_safe` on the command line or in an option file.

To specify the location of the data directory explicitly, use the `--datadir` [385] option. However, normally you can tell `mysqld` the location of the base directory under which MySQL is installed and it looks for the data directory there. You can do this with the `--basedir` [384] option.

To check the effect of specifying path options, invoke `mysqld` with those options followed by the `--verbose` [395] and `--help` [384] options. For example, if you change location into the directory where `mysqld` is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

You can specify other options such as `--datadir` [385] as well, but `--verbose` [395] and `--help` [384] must be the last options. (Prior to MySQL 4.1, omit the `--verbose` [395] option.)

Once you determine the path settings you want, start the server without `--verbose` [395] and `--help` [384].

If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables
```

Or:

```
shell> mysqladmin -h host_name variables
```

host_name is the name of the MySQL server host.

If you get `Errcode 13` (which means `Permission denied`) when starting `mysqld`, this means that the privileges of the data directory or its contents do not permit server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as `root`, but this raises security issues and should be avoided.

On Unix, change location into the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

```
shell> ls -la /usr/local/mysql/var
```

If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

```
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql/var
```

If it possible that even with correct ownership, MySQL may fail to start up if there is other security software running on your system that manages application access to various parts of the file system. In this case, you may need to reconfigure that software to enable `mysqld` to access the directories it uses during normal operation.

If the server fails to start up correctly, check the error log. Log files are located in the data directory (typically `C:\mysql\data` on Windows, `/usr/local/mysql/data` for a Unix binary distribution, and `/usr/local/var` for a Unix source distribution). Look in the data directory for files with names of the form *host_name*`.err` and *host_name*`.log`, where *host_name* is the name of your server host. (Older servers on Windows use `mysql.err` as the error log name.) Then check the last few lines of these files. On Unix, you can use `tail` to display the last few lines:

```
shell> tail host_name.err
shell> tail host_name.log
```

The error log should contain information that indicates why the server could not start. For example, you might see something like this in the log:

```
000729 14:50:10  bdb:  Recovery function for LSN 1 27595 failed
000729 14:50:10  bdb:  warning: ./test/t1.db: No such file or directory
000729 14:50:10  Can't init databases
```

This means that you did not start `mysqld` with the `--bdb-no-recover` [1138] option and Berkeley DB found something wrong with its own log files when it tried to recover your databases. To be able to continue, you should move the old Berkeley DB log files from the database directory to some other place, where you can later examine them. The `BDB` log files are named in sequence beginning with `log.0000000001`, where the number increases over time.

If you are running `mysqld` with `BDB` table support and `mysqld` dumps core at startup, this could be due to problems with the `BDB` recovery log. In this case, you can try starting `mysqld` with `--bdb-no-recover` [1138]. If that helps, you should remove all `BDB` log files from the data directory and try starting `mysqld` again without the `--bdb-no-recover` [1138] option.

If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in Section 5.7, “Running Multiple MySQL Servers on the Same Machine”.)

If no other server is running, try to execute the command `telnet *your_host_name tcp_ip_port_number*`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you do not get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. You will need to track down what program this is and disable it, or else tell `mysqld` to listen to a different port with the `--port` [391] option. In this case, you will also need to specify the port number for client programs when connecting to the server using TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to permit access to the port.

If the server starts but you cannot connect to it, you should make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1       localhost
```

This problem occurs only on systems that do not have a working thread library and for which MySQL must be configured to use MIT-pthreads.

If you cannot get `mysqld` to start, you can try to make a trace file to find the problem by using the `--debug` [385] option. See Section 18.4.3, “The DBUG Package”.

2.10.3 Securing the Initial MySQL Accounts

Part of the MySQL installation process is to set up the `mysql` database that contains the grant tables:

- Windows distributions contain preinitialized grant tables.
- On Unix, the `mysql_install_db` program populates the grant tables. Some installation methods run this program for you. Others require that you execute it manually. For details, see Section 2.10.2, “Unix Postinstallation Procedures”.

The `mysql.user` grant table defines the initial MySQL user accounts and their access privileges:

- Some accounts have the user name `root`. These are superuser accounts that have all privileges and can do anything. The initial `root` account passwords are empty, so anyone can connect to the MySQL server as `root` *without a password* and be granted all privileges.
  - On Windows, prior to MySQL 4.1.10, two `root` accounts are created; one of these is for connections from the local host and the other permits connections from any host. Beginning with MySQL 4.1.10, the Windows installer creates only one `root` account by default, which permits connections only from the local host. If the user selects the **Enable root access from remote machines** option during installation, the Windows installer creates another `root` account that permits connections from any host.
  - On Unix, each `root` account permits connections from the local host. Connections can be made by specifying a host name of `localhost` or the actual host name or IP address.

- Some accounts are for anonymous users. These have an empty user name. The anonymous accounts have no password, so anyone can use them to connect to the MySQL server.
  - On Windows, one anonymous account permits connections from the local host. It has all privileges, just like the `root` accounts. The other permits connections from any host.
  - On Unix, each anonymous account permits connections from the local host. Connections can be made by specifying a host name of `localhost` for one of the accounts, or the actual host name or IP address for the other.

To display which accounts exist in the `mysql.user` table and check whether their passwords are empty, use the following statement:

```
mysql> SELECT User, Host, Password FROM mysql.user;
+------+-------------------+----------+
| User | Host              | Password |
+------+-------------------+----------+
| root | localhost         |          |
| root | myhost.example.com |          |
|      | localhost         |          |
|      | myhost.example.com |          |
+------+-------------------+----------+
```

This output indicates that there are several `root` and anonymous-user accounts, none of which have passwords. The output might differ on your system, but the presence of accounts with empty passwords means that your MySQL installation is unprotected until you do something about it:

- You should assign a password to each MySQL `root` account.
- If you want to prevent clients from connecting as anonymous users without a password, you should either assign a password to each anonymous account or else remove the accounts.

In addition, the `mysql.db` table contains rows that permit all accounts to access the `test` database and other databases with names that start with `test` (on Windows) or that start with `test_` (on Unix). Access to test databases is permitted even for accounts that otherwise have no special privileges such as the default anonymous accounts. This is convenient for testing but inadvisable on production servers. Administrators who want database access restricted only to accounts that have permissions granted explicitly for that purpose should remove these `mysql.db` table rows.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the `root` accounts, then for the anonymous accounts. The instructions also cover how to remove the anonymous accounts, should you prefer not to permit anonymous access at all, and describe how to remove permissive access to test databases. Replace *`newpwd`* in the examples with the password that you want to use. Replace *`host_name`* with the name of the server host. You can determine this name from the output of the preceding `SELECT` statement. For the output shown, *`host_name`* is `myhost.example.com`.

Note

For additional information about setting passwords, see Section 5.6.5, “Assigning Account Passwords”. If you forget your `root` password after setting it, see Section B.5.4.1, “How to Reset the Root Password”.

You might want to defer setting the passwords until later, to avoid the need to specify them while you perform additional setup or testing. However, be sure to set them before using your installation for production purposes.

To set up additional accounts, see Section 5.6.2, “Adding User Accounts”.

Assigning `root` Account Passwords

The `root` account passwords can be set several ways. The following discussion demonstrates three methods:

- Use the `SET PASSWORD` statement
- Use the `UPDATE` statement
- Use the `mysqladmin` command-line client program

To assign passwords using `SET PASSWORD`, connect to the server as `root` and issue a `SET PASSWORD` statement for each `root` account listed in the `mysql.user` table. Be sure to encrypt the password using the `PASSWORD()` [868] function.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

The last statement is unnecessary if the `mysql.user` table has no `root` account with a host value of `%`.

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

You can also use a single statement that assigns a password to all `root` accounts by using `UPDATE` to modify the `mysql.user` table directly. This method works on any platform:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

To assign passwords to the `root` accounts using `mysqladmin`, execute the following commands:

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

Those commands apply both to Windows and to Unix. The double quotation marks around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

If you are using a server from a *very* old version of MySQL, the `mysqladmin` commands to set the password fail with the message `parse error near 'SET password'`. The solution to this problem is to upgrade the server to a newer version of MySQL.

After the `root` passwords have been set, you must supply the appropriate password whenever you connect as `root` to the server. For example, to shut down the server with `mysqladmin`, use this command:

```
shell> mysqladmin -u root -p shutdown
```

```
Enter password: (enter root password here)
```

Assigning Anonymous Account Passwords

The `mysql` commands in the following instructions include a `-p` option based on the assumption that you have set the `root` account passwords using the preceding instructions and must specify that password when connecting to the server.

To assign passwords to the anonymous accounts, connect to the server as `root`, then use either `SET PASSWORD` or `UPDATE`. Be sure to encrypt the password using the `PASSWORD()` [868] function.

To use `SET PASSWORD` on Windows, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'%' = PASSWORD('newpwd');
```

To use `SET PASSWORD` on Unix, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('newpwd');
```

To set the anonymous-user account passwords with `UPDATE`, do this (on any platform):

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

Removing Anonymous Accounts

If you prefer to remove any anonymous accounts rather than assigning them passwords, do so as follows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The `DELETE` statement applies both to Windows and to Unix. On Windows, if you want to remove only the anonymous account that has the same privileges as `root`, do this instead:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

That account permits anonymous access but has full privileges, so removing it improves security.

Securing Test Databases

By default, the `mysql.db` table contains rows that permit access by any user to the `test` database and other databases with names that start with `test_`. (These rows have an empty `User` column value, which

for access-checking purposes matches any user name.) This means that such databases can be used even by accounts that otherwise possess no privileges. If you want to remove any-user access to test databases, do so as follows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DELETE FROM mysql.db WHERE Db LIKE 'test%';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the privilege change remains unnoticed by the server until you restart it.

With the preceding change, only users who have global database privileges or privileges granted explicitly for the `test` database can use it. However, if you do not want the database to exist at all, drop it:

```
mysql> DROP DATABASE test;
```

2.11 Upgrading or Downgrading MySQL

2.11.1 Upgrading MySQL

As a general rule, to upgrade from one release series to another, you should go to the next series rather than skipping a series. For example, if you currently are running MySQL 3.23 and wish to upgrade to a newer series, upgrade to MySQL 4.0 rather than to 4.1 or 5.0.

Whenever you perform an upgrade, use the items in the following checklist as a guide:

- Before any upgrade, back up your databases, including the `mysql` database that contains the grant tables.
- Read *all* the notes the upgrading section for the release series to which you are upgrading. Read the change notes as well. These provide information about new features you can use.
- Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. After you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.
- If you run MySQL Server on Windows, see Section 2.3.14, “Upgrading MySQL on Windows”.
- If you use replication, see Section 14.6, “Upgrading a Replication Setup”, for information on upgrading your replication setup.
- If you upgrade an installation originally produced by installing multiple RPM packages, it is best to upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.
- If you previously installed a MySQL-Max distribution that includes a server named `mysqld-max`, and then upgrade later to a non-Max version of MySQL, `mysqld_safe` still attempts to run the old `mysqld-max` server. If you perform such an upgrade, you should remove the old `mysqld-max` server manually to ensure that `mysqld_safe` runs the new `mysqld` server.
- If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF

with a different nonconflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See Section 8.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

You can always move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL. Before MySQL 4.1, if you change the character set when running MySQL, you must run `myisamchk -r -q --set-character-set=charset_name` on all `MyISAM` tables. Otherwise, your indexes may not be ordered correctly, because changing the character set may also change the sort order. As of MySQL 4.1, to convert tables created before 4.1 to the format that includes character set and collation information, use the instructions in Section 9.1.11.2, “Converting 4.0 Character Columns to 4.1 Format”.

Normally, you can upgrade MySQL to a newer MySQL version without having to do any changes to your tables. Please confirm whether the upgrade notes to the particular version you are upgrading to tell you anything about this. If there would be any incompatibilities you can use `mysqldump` to dump your tables before upgrading. After upgrading, reload the dump file using `mysql` or `mysqlimport` to re-create your tables.

If you are cautious about using new versions, you can always rename your old `mysqld` before installing a newer one. For example, if you are using a version of MySQL 4.0 and want to upgrade to 4.1, rename your current server from `mysqld` to `mysqld-4.0`. If your new `mysqld` then does something unexpected, you can simply shut it down and restart with your old `mysqld`.

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, you should check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example from `libmysqlclient.so.15` to `libmysqlclient.so.16`.

If problems occur, such as that the new `mysqld` server does not want to start or that you cannot connect without a password, verify that you do not have some old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` [235] option (for example, `mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a “dummy” database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the `mysql` database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.

It is a good idea to rebuild and reinstall the Perl `DBD::mysql` module whenever you install a new release of MySQL. The same applies to other MySQL interfaces as well, such as PHP `mysql` and extensions or the Python `MySQLdb` module.

2.11.1.1 Upgrading from MySQL 4.0 to 4.1

Note

It is good practice to back up your data before installing any new version of software. Although MySQL works very hard to ensure a high level of quality, you should protect your data by making a backup.

> To upgrade to 4.1 from any previous version, MySQL recommends that you dump your tables with `mysqldump` before upgrading and reload the dump file after upgrading.

In general, you should do the following when upgrading from MySQL 4.0 to 4.1.

- Read *all* the items in the following sections to see whether any of them might affect your applications:
  - Section 2.11.1, “Upgrading MySQL”, has general update information.
  - The items in the change lists found later in this section enable you to identify upgrade issues that apply to your current MySQL installation.
  - The MySQL 4.1 change history describes significant new features you can use in 4.1 or that differ from those found in MySQL 4.0. Some of these changes may result in incompatibilities. See Section C.1, “Changes in Release 4.1.x (Lifecycle Support Ended)”.

  Note particularly any changes that are marked **Known issue** or **Incompatible change**. These incompatibilities with earlier versions of MySQL may require your attention *before you upgrade*. Note particularly the items under “Server Changes” that related to changes in character set support.
- After upgrading, update the grant tables to obtain the new longer `Password` column that is needed for more secure handling of passwords. The procedure uses `mysql_fix_privilege_tables` and is described in Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”. If you do not do this, MySQL does not use the new more secure protocol to authenticate. Implications of the password-handling change for applications are given later in this section.
- If you run MySQL Server on Windows, see Section 2.3.14, “Upgrading MySQL on Windows”. You should also be aware that two of the Windows MySQL servers were renamed in MySQL 4.1. See Section 2.3.8, “Selecting a MySQL Server Type”.
- If you use replication, see Section 14.6, “Upgrading a Replication Setup”, for information on upgrading your replication setup.
- The Berkeley DB table handler is updated to DB 4.1 (from 3.2) which has a new log format. If you have to downgrade back to 4.0 you must use `mysqldump` to dump your `BDB` tables in text format and delete all `log.XXXXXXXXXX` files before you start MySQL 4.0 and reload the data.
- MySQL 4.1.3 introduces support for per-connection time zones. See Section 9.7, “MySQL Server Time Zone Support”. To enable recognition of named time zones, you should create the time zone tables in the `mysql` database. For instructions, see Section 2.10, “Postinstallation Setup and Testing”.
- If you are using an old `DBD-mysql` module (`Msql-MySQL-modules`) you must upgrade to the newer `DBD-mysql` module. Anything above `DBD-mysql` 2.xx should be satisfactory.

  If you do not upgrade, some methods (such as `DBI->do()`) do not notice error conditions correctly.
- The `--defaults-file=`*`option_file_name`* [235] option gives an error if the option file does not exist.
- Some notes about upgrading from MySQL 4.0 to MySQL 4.1 on Netware: Make sure to upgrade Perl and PHP versions. Download Perl 5 for Netware from http://forge.novell.com/modules/xfmod/project/?perl5 and PHP from http://forge.novell.com/modules/xfmod/project/?php. Download and install the Perl module for MySQL 4.1 from http://forge.novell.com/modules/xfmod/project/showfiles.php?group_id=1126 and the PHP Extension for MySQL 4.1 from http://forge.novell.com/modules/xfmod/project/showfiles.php?group_id=1078.

If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a “dummy” database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the `mysql` database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.

Several visible behaviors have changed between MySQL 4.0 and MySQL 4.1 to fix some critical bugs and make MySQL more compatible with standard SQL. These changes may affect your applications.

Some of the 4.1 behaviors can be tested in 4.0 before performing a full upgrade to 4.1. We have added to later MySQL 4.0 releases (from 4.0.12 on) a `--new` startup option for `mysqld`. See Section 5.1.2, “Server Command Options”.

This option gives you the 4.1 behavior for the most critical changes. You can also enable these behaviors for a given client connection with the `SET @@new=1` command, or turn them off if they are on with `SET @@new=0`.

If you believe that some of the 4.1 changes affect you, we recommend that before upgrading to 4.1, you download the latest MySQL 4.0 version and run it with the `--new` option by adding the following to your config file:

```
[mysqld-4.0]
new
```

That way you can test the new behaviors in 4.0 to make sure that your applications work with them. This helps you have a smooth, painless transition when you perform a full upgrade to 4.1 later. Putting the `--new` option in the `[mysqld-4.0]` option group ensures that you do not accidentally later run the 4.1 version with the `--new` option.

The following lists describe changes that may affect applications and that you should watch out for when upgrading from MySQL 4.0 to 4.1.

Server Changes

The most notable change is that character set support has been improved. The server supports multiple character sets, and all tables and nonbinary string columns (`CHAR`, `VARCHAR`, and `TEXT`) have a character set. See Section 9.1, “Character Set Support”. Binary string columns (`BINARY`, `VARBINARY`, and `BLOB`) contain strings of bytes and do not have a character set.

Note

This change in character set support results in the potential for table damage if you do not upgrade properly, so consider carefully the incompatibilities noted here.

- **Incompatible change**: There are conditions under which you should rebuild tables. In general, to rebuild a table, dump it with `mysqldump` and reload the dump file. Some items in the following list indicate alternatives means for rebuilding.

  - If you have created or used `InnoDB` tables with `TIMESTAMP` columns in MySQL versions 4.1.0 to 4.1.3, you must rebuild those tables when you upgrade to MySQL 4.1.4 or later. The storage format in those MySQL versions for `TIMESTAMP` columns was incorrect. If you upgrade from MySQL 4.0 to 4.1.4 or later, no rebuild of tables with `TIMESTAMP` columns is needed.

  - Starting from MySQL 4.1.3, `InnoDB` uses the same character set comparison functions as MySQL for non-`latin1_swedish_ci` character strings that are not `BINARY`. This changes the sorting order

of space and characters with a code < ASCII(32) in those character sets. For `latin1_swedish_ci` character strings and `BINARY` strings, `InnoDB` uses its own pad-spaces-at-end comparison method, which stays unchanged. Note that `latin1_swedish_ci` is the default collation order for `latin1` in 4.0. If you have an `InnoDB` table created with MySQL 4.1.2 or earlier, with an index on a non-`latin1_swedish_ci` character set and collation order column that is not `BINARY` (in the case of 4.1.0 and 4.1.1, with any character set and collation), and that column may contain characters with a code < ASCII(32), you should do `ALTER TABLE` or `OPTIMIZE TABLE` on it to regenerate the index, after upgrading to MySQL 4.1.3 or later. You can also rebuild the table from a dump.

`MyISAM` tables also have to be rebuilt or repaired in these cases. You can use `mysqldump` to dump them in 4.0 and then reload them in 4.1. An alternative is to use `OPTIMIZE TABLE` after upgrading, but this *must* be done before any updates are made in 4.1.

- As of MySQL 4.1.2, string comparison works according to the SQL standard: Instead of stripping end spaces before comparison, we now extend the shorter string with spaces. The problem with this is that now `'a' > 'a\t'`, which it was not before. If you have any tables where you have indexes on `CHAR`, `VARCHAR` or `TEXT` column in which the last character in index values may be less than `ASCII(32)`, you should rebuild those indexes to ensure that the table is correct.
- If you have used column prefix indexes on UTF-8 columns or other multi-byte character set columns in MySQL 4.1.0 to 4.1.5, you must rebuild the tables when you upgrade to MySQL 4.1.6 or later.
- If you have used accent characters (characters with byte values of 128 to 255) in database names, table names, constraint names, or column names in versions of MySQL earlier than 4.1, you cannot upgrade to MySQL 4.1 directly, because 4.1 uses UTF-8 to store metadata. Use `RENAME TABLE` to overcome this if the accent character is in the table name or the database name, or rebuild the table.
- `MyISAM` tables now use an improved checksum algorithm in MySQL 4.1. If you have `MyISAM` tables with live checksum enabled (you used `CHECKSUM=1` in `CREATE TABLE` or `ALTER TABLE`), these tables appear to be corrupted following an upgrade. Use `REPAIR TABLE` to recalculate the checksum for each such table.

- **Incompatible change**: MySQL interprets length specifications in character column definitions in characters. (Earlier versions interpret them in bytes.) For example, `CHAR(N)` means *`N`* characters, not *`N`* bytes.

  For single-byte character sets, this change makes no difference. However, if you upgrade to MySQL 4.1 and configure the server to use a multi-byte character set, the apparent length of character columns changes. Suppose that a 4.0 table contains a `CHAR(8)` column used to store `ujis` characters. Eight bytes can store from two to four `ujis` characters. If you upgrade to 4.1 and configure the server to use `ujis` as its default character set, the server interprets character column lengths based on the maximum size of a `ujis` character, which is three bytes. The number of three-byte characters that fit in eight bytes is two. Consequently, if you use `SHOW CREATE TABLE` to view the table definition, MySQL displays `CHAR(2)`. You can retrieve existing data from the table, but you can only store new values containing up to two characters. To correct this issue, use `ALTER TABLE` to change the column definition. For example:

  ```
  ALTER TABLE tbl_name MODIFY col_name CHAR(8);
  ```

- **Incompatible change:** As of MySQL 4.1.2, handling of the `FLOAT` and `DOUBLE` floating-point data types is more strict to follow standard SQL. For example, a data type of `FLOAT(3,1)` stores a maximum value of 99.9. Before 4.1.2, the server permitted larger numbers to be stored. That is, it stored a value such as 100.0 as 100.0. As of 4.1.2, the server clips 100.0 to the maximum permissible value of 99.9. If you have tables that were created before MySQL 4.1.2 and that contain floating-point data not strictly legal for the data type, you should alter the data types of those columns. For example:

```
ALTER TABLE tbl_name MODIFY col_name FLOAT(4,1);
```

- **Incompatible change**: In connection with the support for per-connection time zones in MySQL 4.1.3, the `timezone` [432] system variable was renamed to `system_time_zone` [430].

- **Incompatible change**: For `ENUM` columns that had enumeration values containing commas, the commas were mapped to 0xff internally. However, this rendered the commas indistinguishable from true 0xff characters in the values. This no longer occurs. However, the fix requires that you dump and reload any tables that have `ENUM` columns containing true 0xff in their values: Dump the tables using `mysqldump` with the current server before upgrading from a version of MySQL 4.1 older than 4.1.23 to version 4.1.23 or newer.

- **Incompatible change**: The interface to aggregate user-defined functions changed as of MySQL 4.1.1. You must declare a `xxx_clear()` function for each aggregate function `XXX()`. `xxx_clear()` is used instead of `xxx_reset()`. See Section 18.2.2.2, “UDF Calling Sequences for Aggregate Functions”.

- **Incompatible change:** MySQL 4.1 stores table names and column names in `utf8`. If you have table names or column names that use characters outside of the standard 7-bit US-ASCII range, you may have to do a `mysqldump` of your tables in MySQL 4.0 and restore them after upgrading to MySQL 4.1. The symptom for this problem is that you get a `table not found` error when trying to access your tables. In this case, you should be able to downgrade back to MySQL 4.0 and access your data.

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0. That is because, for earlier versions, `InnoDB` is not aware of multiple tablespaces.

- All tables and nonbinary string columns (`CHAR`, `VARCHAR`, and `TEXT`) have a character set. See Section 9.1, “Character Set Support”. Binary string columns (`BINARY`, `VARBINARY`, and `BLOB`) contain strings of bytes and do not have a character set.

  Character set information is displayed by `SHOW CREATE TABLE` and `mysqldump`. (MySQL versions 4.0.6 and above can read the new dump files; older versions cannot.) This change should not affect applications that use only one character set.

- If you were using columns with the `CHAR BINARY` or `VARCHAR BINARY` data types in MySQL 4.0, these were treated as binary strings. To have them treated as binary strings in MySQL 4.1, you should convert them to the `BINARY` and `VARBINARY` data types, respectively.

- If you have table columns that store character data represented in a character set that the 4.1 server supports directly, you can convert the columns to the proper character set using the instructions in Section 9.1.11.2, “Converting 4.0 Character Columns to 4.1 Format”. Also, database, table, and column identifiers are stored internally using Unicode (UTF-8) regardless of the default character set. See Section 8.2, “Database, Table, Index, Column, and Alias Names”.

- The table definition format used in `.frm` files has changed slightly in 4.1. MySQL 4.0 versions from 4.0.11 on can read the new `.frm` format directly, but older versions cannot. If you need to move tables from 4.1 to a version earlier than 4.0.11, you should use `mysqldump`.

- Windows servers support connections from local clients using shared memory if run with the `--shared-memory` [392] option. If you are running multiple servers this way on the same Windows machine, you should use a different `--shared-memory-base-name` option for each server.

- As of MySQL 4.1.21, the `lc_time_names` [417] system variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()` [832], `DAYNAME()` [833] and `MONTHNAME()` [837] functions. See Section 9.8, “MySQL Server Locale Support”.

- As of MySQL 4.1.10a, the server by default no longer loads user-defined functions (UDFs) unless they have at least one auxiliary symbol defined in addition to the main function symbol. This behavior can be overridden with the `--allow-suspicious-udfs` [384] option. See Section 18.2.2.6, "User-Defined Function Security Precautions".

Client Changes

- As of MySQL 4.1, `mysqldump` has the `--opt` [297] and `--quote-names` [298] options enabled by default. You can turn these off using `--skip-opt` [299] and `--skip-quote-names`.

SQL Changes

- **Incompatible change**: `TIMESTAMP` is returned in MySQL 4.1 as a string in `'YYYY-MM-DD HH:MM:SS'` format. (See Section 10.3.1.2, "`TIMESTAMP` Properties as of MySQL 4.1".) From 4.0.12 on, the `--new` option can be used to make a 4.0 server behave as 4.1 in this respect. The effect of this option is described in Section 10.3.1.1, "`TIMESTAMP` Properties Prior to MySQL 4.1".

  When running the server with `--new`, if you want to have a `TIMESTAMP` column returned as a number (as MySQL 4.0 does by default), you should add `+0` when you retrieve it:

```
mysql> SELECT ts_col + 0 FROM tbl_name;
```

  Display widths for `TIMESTAMP` columns are no longer supported in MySQL 4.1. For example, if you declare a column as `TIMESTAMP(10)`, the `(10)` is ignored.

- **Incompatible change**: Binary values such as `0xFFDF` are assumed to be strings instead of numbers. This fixes some problems with character sets where it is convenient to input a string as a binary value. With this change, you should use `CAST()` [859] if you want to compare binary values numerically as integers:

```
mysql> SELECT CAST(0xFEFF AS UNSIGNED INTEGER)
    ->        < CAST(0xFF AS UNSIGNED INTEGER);
        -> 0
```

  If you do not use `CAST()` [859], a lexical string comparison is made instead:

```
mysql> SELECT 0xFEFF < 0xFF;
        -> 1
```

  Using binary items in a numeric context or comparing them using the `=` operator should work as before. (The `--new` option can be used from 4.0.13 on to make a 4.0 server behave as 4.1 in this respect.)

- **Incompatible change:** Before MySQL 4.1.13, conversion of `DATETIME` values to numeric form by adding zero produced a result in `YYYYMMDDHHMMSS` format. The result of `DATETIME+0` is now in `YYYYMMDDHHMMSS.000000` format.

- **Incompatible change:** In MySQL 4.1.12, the behavior of `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` has changed when the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values both are empty. Formerly, a column was read or written using the display width of the column. For example, `INT(4)` was read or written using a field with a width of 4. Now columns are read and written using a field width wide enough to hold all values in the field. However, data files written before this change was made might not be reloaded correctly with `LOAD DATA INFILE` for MySQL 4.1.12 and up. This change also affects data files read by `mysqlimport` and written by `mysqldump --tab`, which use `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`. For more information, see Section 12.2.5, "`LOAD DATA INFILE` Syntax".

- **Incompatible change**: Before MySQL 4.1.1, the statement parser was less strict and its string-to-date conversion would ignore everything up to the first digit. As a result, invalid statements such as the following were accepted:

```
INSERT INTO t (datetime_col) VALUES ('stuff 2005-02-11 10:17:01');
```

  As of MySQL 4.1.1, the parser is stricter and treats the string as an invalid date, so the preceding statement results in a warning.

- **Incompatible change**: In MySQL 4.1.2, the `Type` column in the output from `SHOW TABLE STATUS` was renamed to `Engine`. This affects applications that identify output columns by name rather than by position.

- **Incompatible change**: The syntax for multiple-table `DELETE` statements that use table aliases changed between MySQL 4.0 and 4.1. In MySQL 4.0, you should use the true table name to refer to any table from which rows should be deleted:

```
DELETE test FROM test AS t1, test2 WHERE ...
```

  In MySQL 4.1, you must use the alias:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

  We did not make this change in 4.0 to avoid breaking any old 4.0 applications that were using the old syntax. However, if you use such `DELETE` statements and are using replication, the change in syntax means that a 4.0 master cannot replicate to 4.1 (or higher) slaves.

- Some keywords are reserved in MySQL 4.1 that were not reserved in MySQL 4.0. See Section 8.3, “Reserved Words”.

- The `LOAD DATA FROM MASTER` and `LOAD TABLE FROM MASTER` statements are deprecated. See Section 12.5.2.2, “`LOAD DATA FROM MASTER` Syntax”, for recommended alternatives.

- For functions that produce a `DATE`, `DATETIME`, or `TIME` value, the result returned to the client is fixed up to have a temporal type. For example, in MySQL 4.1, you obtain the following:

```
mysql> SELECT CAST('2001-1-1' AS DATETIME);
       -> '2001-01-01 00:00:00'
```

  In MySQL 4.0, the result of the stement is different:

```
mysql> SELECT CAST('2001-1-1' AS DATETIME);
       -> '2001-01-01'
```

- `DEFAULT` values no longer can be specified for `AUTO_INCREMENT` columns. (In 4.0, a `DEFAULT` value is silently ignored; in 4.1, an error occurs.)

- `LIMIT` no longer accepts negative arguments. Use some large number (maximum 18446744073709551615) instead of -1.

- `SERIALIZE` is no longer a valid mode value for the `sql_mode` [429] variable. You should use `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` instead. `SERIALIZE` is no longer valid for the `--sql-mode` [394] option for `mysqld`, either. Use `--transaction-isolation=SERIALIZABLE` instead.

- A new startup option named `innodb_table_locks` was added that causes `LOCK TABLE` to also acquire `InnoDB` table locks. This option is enabled by default. This can cause deadlocks in applications that use `autocommit = 1` [406] and `LOCK TABLES`. If you application encounters deadlocks after upgrading, you may need to add `innodb_table_locks = 0` to your `my.cnf` file.

C API Changes

- **Incompatible change**: The `mysql_shutdown()` C API function has an extra parameter as of MySQL 4.1.3: `SHUTDOWN`-level. You should convert any `mysql_shutdown(X)` call you have in your application to `mysql_shutdown(X,SHUTDOWN_DEFAULT)`. Any third-party API that links against the C API library must be modified to account for this change or it will not compile.

- Some C API calls such as `mysql_real_query()` return `1` on error, not `-1`. You may have to change some old applications if they use constructs like this:

```
if (mysql_real_query(mysql_object, query, query_length) == -1)
{
  printf("Got error");
}
```

  Change the call to test for a nonzero value instead:

```
if (mysql_real_query(mysql_object, query, query_length) != 0)
{
  printf("Got error");
}
```

Password-Handling Changes

The password hashing mechanism changed in 4.1 to provide better security; this may cause compatibility problems if you have clients using the client library from 4.0 or earlier. (It is very likely that you have 4.0 clients in situations where clients connect from remote hosts that have not yet upgraded to 4.1.) The following list indicates some possible upgrade strategies. They represent various tradeoffs between the goals of compatibility with old clients and security.

- Only upgrade the client to use 4.1 client libraries (not the server). No behavior changes (except the return value of some API calls), but you cannot use any of the new features provided by the 4.1 client/server protocol, either. (MySQL 4.1 has an extended client/server protocol that offers such features as prepared statements and multiple result sets.) See Section 17.6.7, “C API Prepared Statements”.

- Upgrade to 4.1 and run the `mysql_fix_privilege_tables` script to widen the `Password` column in the `user` table so that it can hold long password hashes. However—to provide backward compatibility enabling pre-4.1 clients to continue connecting to their short-hash accounts—run the server with the `--old-passwords` [391] option. Eventually, when all your clients are upgraded to 4.1, you can stop using the `--old-passwords` [391] server option. You can also change the passwords for your MySQL accounts to use the new more secure format. A 4.1 installation using only the improved authentication protocol is the most secure one.

Further background on password hashing with respect to client authentication and password-changing operations may be found in Section 5.4.2.3, “Password Hashing in MySQL”, and Section B.5.2.4, “`Client does not support authentication protocol`”.

2.11.1.2 Upgrading from MySQL 3.23 to 4.0

In general, you should do the following when upgrading from MySQL 3.23 to 4.0:

- Read all the items in Section 2.11.1, “Upgrading MySQL”, to see whether any of them might affect your applications.
- Read all the items in the change list found later in this section to see whether any of them might affect your applications. Note particularly any that are marked **Known issue** or **Incompatible change**; these result in incompatibilities with earlier versions of MySQL.
- Read the 4.0 changelog to see what significant new features you can use in 4.0. See Section C.2, “Changes in Release 4.0.x (Lifecycle Support Ended)”.
- If you run MySQL Server on Windows, see Section 2.3.14, “Upgrading MySQL on Windows”.
- After upgrading, update the grant tables to add new privileges and features. This procedure uses the `mysql_fix_privilege_tables` script and is described in Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.
- If you use replication, see Section 14.6, “Upgrading a Replication Setup”, for information on upgrading your replication setup.
- Edit any MySQL startup scripts or option files so that they do not use any of the options described as deprecated later in this section.
- Convert your old `ISAM` tables to `MyISAM` format. One way to do this is with the `mysql_convert_table_format` script. (This is a Perl script; it requires that `DBI` be installed.) To convert all of the tables in a given database, use this command:

  ```
  shell> mysql_convert_table_format database db_name
  ```

  Note that the above command should be used only if *all* tables in the database are `ISAM` or `MyISAM` tables. To avoid converting tables of other types to `MyISAM`, you can explicitly list the names of the `ISAM` tables following the database name on the command line.

  Individual tables can be changed to `MyISAM` by using the following `ALTER TABLE` statement for each table to be converted:

  ```
  mysql> ALTER TABLE tbl_name TYPE=MyISAM;
  ```

  If you are not sure of the storage engine for a given table, use this statement:

  ```
  mysql> SHOW TABLE STATUS LIKE 'tbl_name';
  ```

- Ensure that you do not have any MySQL clients that use shared libraries (like the Perl `DBD::mysql` module). If you do, you should recompile them, because the data structures used in `libmysqlclient.so` have changed. The same applies to other MySQL interfaces such as the Python `MySQLdb` module.

MySQL 4.0 works even if you do not perform the preceding actions, but you cannot use the new security privileges in MySQL 4.0 and you may run into problems when upgrading later to MySQL 4.1 or newer. The `ISAM` file format still works in MySQL 4.0, but is deprecated and is not compiled in by default as of MySQL 4.1. `MyISAM` tables should be used instead.

Old clients should work with a MySQL 4.0 server without any problems.

Even if you perform the preceding actions, you can still downgrade to MySQL 3.23.52 or newer if you run into problems with the MySQL 4.0 series. In this case, you must use `mysqldump` to dump any tables that

use full-text indexes and reload the dump file into the 3.23 server. This is necessary because 4.0 uses an improved format for full-text indexing that is not backward-compatible.

The following lists describe changes that may affect applications and that you should watch out for when upgrading from MySQL 3.23 to 4.0.

Server Changes

- As of MySQL 4.0.24, the server by default no longer loads user-defined functions unless they have at least one auxiliary symbol defined in addition to the main function symbol. This behavior can be overridden with the `--allow-suspicious-udfs` [384] option. See Section 18.2.2.6, "User-Defined Function Security Precautions".

- MySQL 4.0 has many new privileges in the `mysql.user` table. See Section 5.5.1, "Privileges Provided by MySQL".

  For these new privileges to work, you must update the grant tables. The procedure for this is described in Section 4.4.5, "`mysql_fix_privilege_tables` — Upgrade MySQL System Tables". Until you do this, all accounts have the `SHOW DATABASES` [492], `CREATE TEMPORARY TABLES` [491], and `LOCK TABLES` [492] privileges. `SUPER` [493] and `EXECUTE` [491] privileges take their value from `PROCESS` [492]. `REPLICATION SLAVE` [492] and `REPLICATION CLIENT` [492] take their values from `FILE` [491].

  If you have any scripts that create new MySQL user accounts, you may want to change them to use the new privileges. If you are not using `GRANT` commands in the scripts, this is a good time to change your scripts to use `GRANT` instead of modifying the grant tables directly.

  From version 4.0.2 on, the option `--safe-show-database` [391] is deprecated (and no longer does anything). See Section 5.4.4, "Security-Related `mysqld` Options".

  If you get `Access denied` errors for new users in version 4.0.2 and up, you should check whether you need some of the new grants that you did not need before. In particular, you need `REPLICATION SLAVE` [492] (instead of `FILE` [491]) for new slave servers.

- `safe_mysqld` has been renamed to `mysqld_safe`. For backward compatibility, binary distributions will for some time include `safe_mysqld` as a symlink to `mysqld_safe`.

- `InnoDB` support is included by default in binary distributions. If you build MySQL from source, `InnoDB` is configured in by default. If you do not use `InnoDB` and want to save memory when running a server that has `InnoDB` support enabled, use the `--skip-innodb` [1066] server startup option. To compile MySQL without `InnoDB` support, run `configure` with the `--without-innodb` option.

- Values for the startup parameters `myisam_max_extra_sort_file_size` [421] and `myisam_max_extra_sort_file_size` [421] are given in bytes (prior to 4.0.3,they were given in megabytes).

- `mysqld` has the option `--temp-pool` [394] enabled by default because this gives better performance with some operating systems (most notably Linux).

- The `mysqld` startup options `--skip-locking` and `--enable-locking` were renamed to `--skip-external-locking` [392] and `--external-locking` [387]. `--skip-locking` and `--enable-locking` are deprecated.

- External system locking of `MyISAM`/`ISAM` files is turned off by default. You can turn this on with `--external-locking` [387]. (However, this is never needed for most users.)

- The following startup variables and options were renamed:

Name in 3.23	Name in 4.0 (and above)
myisam_bulk_insert_tree_size	bulk_insert_buffer_size [408]
query_cache_startup_type	query_cache_type [424]
record_buffer	read_buffer_size [425]
record_rnd_buffer	read_rnd_buffer_size [426]
sort_buffer	sort_buffer_size [427]
--warnings	--log-warnings [389]
--err-log	--log-error [388] (for mysqld_safe)

The startup options record_buffer, sort_buffer, and warnings still work in MySQL 4.0 but are deprecated.

SQL Changes

- Some keywords are reserved in MySQL 4.0 that were not reserved in MySQL 3.23. See Section 8.3, “Reserved Words”.
- The following SQL variables have been renamed:

Name in 3.23	Name in 4.0 and above
sql_big_tables	big_tables [407]
sql_low_priority_updates	low_priority_updates [418]
sql_max_join_size	max_join_size [419]
sql_query_cache_type	query_cache_type [424]

The older names still work in MySQL 4.0 but are deprecated.

- You must use SET GLOBAL SQL_SLAVE_SKIP_COUNTER=skip_count instead of SET SQL_SLAVE_SKIP_COUNTER=skip_count.
- SHOW MASTER STATUS returns an empty set if binary logging is not enabled.
- SHOW SLAVE STATUS returns an empty set if the slave is not initialized.
- SHOW INDEX has two more columns in 4.0 than in 3.23 (Null and Index_type).
- The format of SHOW OPEN TABLES changed.
- As of MySQL 4.0.11, ORDER BY col_name DESC sorts NULL values last. In 3.23 and in earlier 4.0 versions, this was not always consistent.
- CHECK, LOCALTIME, and LOCALTIMESTAMP are reserved words.
- DOUBLE and FLOAT columns honor the UNSIGNED flag on storage (previously, UNSIGNED was ignored for these columns).
- The result of all bitwise operators (|, &, <<, >>, and ~) is unsigned. This may cause problems if you are using them in a context where you want a signed result. See Section 11.10, “Cast Functions and Operators”.

Note

When you use subtraction between integer values where one is of type `UNSIGNED`, the result is unsigned. In other words, before upgrading to MySQL 4.0, you should check your application for cases in which you are subtracting a value from an unsigned entity and want a negative answer or subtracting an unsigned value from an integer column. You can disable this behavior by using the `--sql-mode=NO_UNSIGNED_SUBTRACTION` [394] option when starting `mysqld`. See Section 5.1.6, “Server SQL Modes”.

- You should use integers to store values in `BIGINT` columns (instead of using strings as in MySQL 3.23). Using strings still works, but using integers is more efficient.
- In MySQL 3.23, `INSERT INTO ... SELECT` always had `IGNORE` enabled. As of 4.0.1, MySQL stops (and possibly rolls back) by default in case of an error unless you specify `IGNORE`.
- You should use `TRUNCATE TABLE` when you want to delete all rows from a table and you do not need to obtain a count of the number of rows that were deleted. (`DELETE FROM` *`tbl_name`* returns a row count in 4.0 and does not reset the `AUTO_INCREMENT` counter, and `TRUNCATE TABLE` is faster.)
- You get an error if you have an active transaction or `LOCK TABLES` statement when trying to execute `TRUNCATE TABLE` or `DROP DATABASE`.
- To use `MATCH ... AGAINST (... IN BOOLEAN MODE)` full-text searches, you must rebuild existing table indexes using `REPAIR TABLE` *`tbl_name`* `USE_FRM`. If you attempt a boolean full-text search without rebuilding the indexes in this manner, the search returns incorrect results. See Section 11.9.6, “Fine-Tuning MySQL Full-Text Search”.
- `LOCATE()` [798] and `INSTR()` [797] are case sensitive if one of the arguments is a binary string. Otherwise they are case insensitive.
- `STRCMP()` [807] uses the current character set when performing comparisons. This makes the default comparison behavior not case sensitive unless one or both of the operands are binary strings.
- `HEX(`*`str`*`)` [796] returns the characters in *`str`* converted to hexadecimal. If you want to convert a number to hexadecimal, you should ensure that you call `HEX()` [796] with a numeric argument.
- `RAND(seed)` [822] returns a different random number series in 4.0 than in 3.23; this was done to further differentiate `RAND(seed)` [822] and `RAND(seed+1)` [822].
- The default type returned by `IFNULL(A,B)` [791] is set to be the more “general” of the types of `A` and `B`. (The general-to-specific order is string, `REAL`, `INTEGER`).

C API Changes

- The old C API functions `mysql_drop_db()`, `mysql_create_db()`, and `mysql_connect()` are no longer supported in MySQL 4.0 unless MySQL is compiled with `CFLAGS=-DUSE_OLD_FUNCTIONS`. It is preferable to change client programs to use the new 4.0 API instead.
- In the `MYSQL_FIELD` structure, `length` and `max_length` have changed from `unsigned int` to `unsigned long`. This should not cause any problems, except that they may generate warning messages when used as arguments in the `printf()` class of functions.
- Multi-threaded clients should use `mysql_thread_init()` and `mysql_thread_end()`. See Section 17.6.3.2, “Writing C API Threaded Client Programs”.

Other Changes

- If you want to recompile the Perl `DBD::mysql` module, use a recent version. Version 2.9003 is recommended. Versions older than 1.2218 should not be used because they use the deprecated `mysql_drop_db()` call.

2.11.2 Downgrading MySQL

This section describes what you should do to downgrade to an older MySQL version in the unlikely case that the previous version worked better than the new one.

If you are downgrading within the same release series (for example, from 4.0.20 to 4.0.19) the general rule is that you merely need to install the new binaries on top of the old ones. There is no need to do anything with the databases. As always, however, it is always a good idea to make a backup.

The following items form a checklist of things you should do whenever you perform a downgrade:

- Read the upgrading section for the release series from which you are downgrading to be sure that it does not have any features you really need. See Section 2.11.1, “Upgrading MySQL”.
- If there is a downgrading section for that version, please read it, too!
- To see which new features were added between the version to which you are downgrading and your current version, see the change logs (Appendix C, *MySQL Release Notes*).
- Check Section 2.11.3, “Checking Whether Tables or Indexes Must Be Rebuilt”, to see whether changes to table formats or to character sets or collations were made between your current version of MySQL and the version to which you are downgrading. If so and these changes result in an incompatibility between MySQL versions, you will need to downgrade the affected tables using the instructions in Section 2.11.4, “Rebuilding or Repairing Tables or Indexes”.

You can always move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL.

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, use `mysqldump` to dump your tables before downgrading. After downgrading, reload the dump file using `mysql` or `mysqlimport` to re-create your tables. For examples, ee Section 2.11.5, “Copying MySQL Databases to Another Machine”.

A typical symptom of a downward-incompatible table format change when you downgrade is that you cannot open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are downgrading to.
2. Restart the newer MySQL server you are downgrading from.
3. Dump any tables that were inaccessible to the older server by using `mysqldump` to create a dump file.
4. Stop the newer MySQL server and restart the older one.
5. Reload the dump file into the older server. Your tables should be accessible.

It might also be the case that the structure of the system tables in the `mysql` database has changed and that downgrading introduces some loss of functionality or requires some adjustments. Here are some examples:

- Trigger creation requires the `TRIGGER` privilege as of MySQL 5.1. In MySQL 5.0, there is no `TRIGGER` privilege and `SUPER` is required instead. If you downgrade from MySQL 5.1 to 5.0, you will need to give the `SUPER` privilege to those accounts that had the `TRIGGER` privilege in 5.1.

- Triggers were added in MySQL 5.0, so if you downgrade from 5.0 to 4.1, you cannot use triggers at all.

2.11.2.1 Downgrading to MySQL 4.0

The table format in 4.1 changed to include more and new character set information. Because of this, you must use `mysqldump` to dump any tables you have created with the newer MySQL server. For example, if all the tables in a particular database need to be dumped to be reverted back to MySQL 4.0 format, use this command:

```
shell> mysqldump --create-options --compatible=mysql40 db_name > dump_file
```

Then stop the newer server, restart the older server, and read in the dump file:

```
shell> mysql db_name < dump_file
```

In the special case that you are downgrading `MyISAM` tables, no special treatment is necessary if all columns in the tables contain only numeric columns or string columns (`CHAR`, `VARCHAR`, `TEXT`, and so forth) that contain only `latin1` data. Your 4.1 tables should be directly usable with a 4.0 server.

If you used the `mysql_fix_privilege_tables` script to upgrade the grant tables, you can either use the preceding method to convert them to back to MySQL 4.0 or do the following in MySQL 4.1 (or above):

```
ALTER TABLE mysql.user
  CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.db
  CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.host
  CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.tables_priv
  CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.columns_priv
  CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.func
  CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

2.11.3 Checking Whether Tables or Indexes Must Be Rebuilt

A binary upgrade or downgrade is one that installs one version of MySQL “in place” over an existing version, without dumping and reloading tables:

1. Stop the server for the existing version if it is running.
2. Install a different version of MySQL. This is an upgrade if the new version is higher than the original version, a downgrade if the version is lower.
3. Start the server for the new version.

In many cases, the tables from the previous version of MySQL can be used without problem by the new version. However, sometimes changes occur that require tables or table indexes to be rebuilt, as described in this section. If you have tables that are affected by any of the issues described here, rebuild the tables or indexes as necessary using the instructions given in Section 2.11.4, “Rebuilding or Repairing Tables or Indexes”.

Index Incompatibilities

If you perform a binary upgrade without dumping and reloading tables, you cannot upgrade directly from MySQL 4.1 to 5.1 or higher. This occurs due to an incompatible change in the `MyISAM` table index format

in MySQL 5.0. Upgrade from MySQL 4.1 to 5.0 and repair all `MyISAM` tables. Then upgrade from MySQL 5.0 to 5.1 and check and repair your tables.

Modifications to the handling of character sets or collations might change the character sort order, which causes the ordering of entries in any index that uses an affected character set or collation to be incorrect. Such changes result in several possible problems:

- Comparison results that differ from previous results
- Inability to find some index values due to misordered index entries
- Misordered `ORDER BY` results
- Tables that `CHECK TABLE` reports as being in need of repair

The solution to these problems is to rebuild any indexes that use an affected character set or collation, either by dropping and re-creating the indexes, or by dumping and reloading the entire table. For information about rebuilding indexes, see Section 2.11.4, “Rebuilding or Repairing Tables or Indexes”.

To check whether a table has indexes that must be rebuilt, consult the following list. It indicates which versions of MySQL introduced character set or collation changes that require indexes to be rebuilt. Each entry indicates the version in which the change occurred and the character sets or collations that the change affects. If the change is associated with a particular bug report, the bug number is given.

The list applies both for binary upgrades and downgrades. For example, Bug #27877 was fixed in MySQL 5.1.24 and 5.4.0, so it applies to upgrades from versions older than 5.1.24 to 5.1.24 or newer, and to downgrades from 5.1.24 or newer to versions older than 5.1.24.

Changes that cause index rebuilding to be necessary:

- MySQL 4.1.2 (Bug #3152)

  String comparison works according to the SQL standard: Instead of stripping end spaces before comparison, we now extend the shorter string with spaces. The problem with this is that now `'a' > 'a\t'`, which it was not before. If you have any tables where you have indexes on `CHAR`, `VARCHAR` or `TEXT` column in which the last character in index values may be less than `ASCII(32)`, you should rebuild those indexes.

- MySQL 4.1.3

  `InnoDB` uses the same character set comparison functions as MySQL for non-`latin1_swedish_ci` character strings that are not `BINARY`. This changes the sorting order of space and characters with a code < ASCII(32) in those character sets. This affects `InnoDB` tables with an index on a non-`latin1_swedish_ci` character set and collation order column that is not `BINARY` if that column contains characters with a code < ASCII(32). (For MySQL 4.1.0 and 4.1.1, it affects indexes with any character set and collation).

- MySQL 5.0.48, 5.1.21 (Bug #29461)

  Affects indexes for columns that use any of these character sets: `eucjpms`, `euc_kr`, `gb2312`, `latin7`, `macce`, `ujis`

- MySQL 5.0.48, 5.1.23 (Bug #27562)

  Affects indexes that use the `ascii_general_ci` collation for columns that contain any of these characters: `'``'` GRAVE ACCENT, `'['` LEFT SQUARE BRACKET, `'\'` REVERSE SOLIDUS, `']'` RIGHT SQUARE BRACKET, `'~'` TILDE

- MySQL 5.1.24, 5.4.0 (Bug #27877)

  Affects indexes that use the `utf8_general_ci` or `ucs2_general_ci` collation for columns that contain `'ß'` LATIN SMALL LETTER SHARP S (German).

2.11.4 Rebuilding or Repairing Tables or Indexes

This section describes how to rebuild a table. This can be necessitated by changes to MySQL such as how data types are handled or changes to character set handling. For example, an error in a collation might have been corrected, necessitating a table rebuild to update the indexes for character columns that use the collation. (For examples, see Section 2.11.3, “Checking Whether Tables or Indexes Must Be Rebuilt”.) It might also be that a table repair or upgrade should be done as indicated by a table check operation such as that performed by `CHECK TABLE` or `mysqlcheck`.

Methods for rebuilding a table include dumping and reloading it, or using `ALTER TABLE` or `REPAIR TABLE`.

Note

If you are rebuilding tables because a different version of MySQL will not handle them after a binary (in-place) upgrade or downgrade, you must use the dump-and-reload method. Dump the tables *before* upgrading or downgrading using your original version of MySQL. Then reload the tables *after* upgrading or downgrading.

If you use the dump-and-reload method of rebuilding tables only for the purpose of rebuilding indexes, you can perform the dump either before or after upgrading or downgrading. Reloading still must be done afterward.

To rebuild a table by dumping and reloading it, use `mysqldump` to create a dump file and `mysql` to reload the file:

```
shell> mysqldump db_name t1 > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all the tables in a single database, specify the database name without any following table name:

```
shell> mysqldump db_name > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all tables in all databases, use the `--all-databases` [293] option:

```
shell> mysqldump --all-databases > dump.sql
shell> mysql < dump.sql
```

Before MySQL 4.1, use the `--opt` [297] and `--quote-names` [298] options. As of 4.1, those options are enabled by default.

To rebuild a table with `ALTER TABLE`, use a “null” alteration; that is, an `ALTER TABLE` statement that “changes” the table to use the storage engine that it already has. For example, if `t1` is a `MyISAM` table, use this statement:

```
mysql> ALTER TABLE t1 ENGINE = MyISAM;
```

If you are not sure which storage engine to specify in the `ALTER TABLE` statement, use `SHOW CREATE TABLE` to display the table definition.

If you must rebuild a table because a table checking operation indicates that the table is corrupt or needs an upgrade, you can use `REPAIR TABLE` if that statement supports the table's storage engine. For example, to repair a `MyISAM` table, use this statement:

```
mysql> REPAIR TABLE t1;
```

For storage engines such as `InnoDB` that `REPAIR TABLE` does not support, use `mysqldump` to create a dump file and `mysql` to reload the file, as described earlier.

For specifics about which storage engines `REPAIR TABLE` supports, see Section 12.4.2.6, "`REPAIR TABLE` Syntax".

`mysqlcheck --repair` provides command-line access to the `REPAIR TABLE` statement. This can be a more convenient means of repairing tables because you can use the `--databases` [285] or `--all-databases` [285] option to repair all tables in specific databases or all databases, respectively:

```
shell> mysqlcheck --repair --databases db_name ...
shell> mysqlcheck --repair --all-databases
```

2.11.5 Copying MySQL Databases to Another Machine

If you are using MySQL 3.23 or later, you can copy the `.frm`, `.MYI`, and `.MYD` files for `MyISAM` tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See Section 13.1, "The `MyISAM` Storage Engine".

The MySQL `ISAM` data and index files (`.ISD` and `*.ISM`, respectively) are dependent upon the architecture and, in some cases, the operating system. If you want to move applications to another machine having a different architecture or operating system than that of the current machine, you should not try to move a database by simply copying the files to the other machine. Use `mysqldump` instead.

By default, `mysqldump` creates a file containing SQL statements. You can then transfer the file to the other machine and use it as input to the `mysql` client.

Try `mysqldump --help` to see what options are available. Before MySQL 4.1, if you are moving the data to a newer version of MySQL, you should add the `--opt` [297] option to the `mysqldump` commands shown here, to take advantage of any optimizations that result in a dump file that is smaller and can be processed faster. (`--opt` [297] is enabled by default as of MySQL 4.1.)

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.gz | mysql db_name
```

You can also use `mysqldump` and `mysqlimport` to transfer the database. For very large tables, this is much faster than simply using `mysqldump`. In the following commands, *`DUMPDIR`* represents the full path name of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the *`DUMPDIR`* directory to a directory on the target machine and load the files into MySQL there:

```
shell> mysqladmin create db_name           # create database
shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Do not forget to copy the `mysql` database because that is where the `user`, `db`, and `host` grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

2.12 Operating System-Specific Notes

2.12.1 Linux Notes

This section discusses issues that have been found to occur on Linux. The first few subsections describe general operating system-related issues, problems that can occur when using binary or source distributions, and postinstallation issues. The remaining subsections discuss problems that occur with Linux on specific platforms.

Note that most of these problems occur on older versions of Linux. If you are running a recent version, you may see none of them.

2.12.1.1 Linux Operating System Notes

MySQL needs at least Linux version 2.0.

Warning

We have seen some strange problems with Linux 2.2.14 and MySQL on SMP systems. Some MySQL users have also reported that they have encountered serious stability problems using MySQL with kernel 2.2.14. If you are using this kernel, you should upgrade to 2.2.19 (or newer) or to a 2.4 or 2.6 kernel. If you have a multiple-CPU machine, you should seriously consider using 2.4 or 2.6 because it gives you a significant speed boost. Your system should also be more stable.

When using LinuxThreads, you should see a minimum of three `mysqld` processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

2.12.1.2 Linux Binary Distribution Notes

The Linux-Intel binary and RPM releases of MySQL are configured for the highest possible speed. We are always trying to use the fastest stable compiler available.

The binary release is linked with `-static`, which means you do not normally need to worry about which version of the system libraries you have. You need not install LinuxThreads, either. A program linked with `-static` is slightly larger than a dynamically linked program, but also slightly faster (3% to 5%). However, one problem with a statically linked program is that you cannot use user-defined functions (UDFs). If you are going to write or use UDFs (this is something for C or C++ programmers only), you must compile MySQL yourself using dynamic linking.

A known issue with binary distributions is that on older Linux systems that use `libc` (such as Red Hat 4.x or Slackware), you get some nonfatal problems with host name resolution. If your system uses `libc` rather than `glibc2`, you probably will encounter some difficulties with host name resolution and `getpwnam()`. This happens because `glibc` unfortunately depends on some external libraries to implement host name resolution and `getpwent()`, even when compiled with `-static`. These problems manifest themselves in two ways:

- You may see the following error message when you run `mysql_install_db`:

  ```
  Sorry, the host 'xxxx' could not be looked up
  ```

  You can deal with this by executing `mysql_install_db --force`, which does not execute the `resolveip` test in `mysql_install_db`. The downside is that you cannot use host names in the grant tables: Except for `localhost`, you must use IP addresses instead. If you are using an old version of MySQL that does not support `--force`, you must manually remove the `resolveip` test in `mysql_install_db` using a text editor.

- You also may see the following error when you try to run `mysqld` with the `--user` [395] option:

  ```
  getpwnam: No such file or directory
  ```

  To work around this problem, start `mysqld` by using the `su` command rather than by specifying the `--user` [395] option. This causes the system itself to change the user ID of the `mysqld` process so that `mysqld` need not do so.

Another solution, which solves both problems, is to not use a binary distribution. Get a MySQL source distribution (in RPM or `.tar.gz` format) and install that instead.

On some Linux 2.2 versions, you may get the error `Resource temporarily unavailable` when clients make a lot of new connections to a `mysqld` server over TCP/IP. The problem is that Linux has a delay between the time that you close a TCP/IP socket and the time that the system actually frees it. There is room for only a finite number of TCP/IP slots, so you encounter the resource-unavailable error if clients attempt too many new TCP/IP connections during a short time. For example, you may see the error when you run the MySQL `test-connect` benchmark over TCP/IP.

We have inquired about this problem a few times on different Linux mailing lists but have never been able to find a suitable resolution. The only known “fix” is for the clients to use persistent connections, or, if you are running the database server and clients on the same machine, to use Unix socket file connections rather than TCP/IP connections.

2.12.1.3 Linux Source Distribution Notes

The following notes regarding `glibc` apply only to the situation when you build MySQL yourself. If you are running Linux on an x86 machine, in most cases it is much better for you to just use our binary. We link our

binaries against the best patched version of `glibc` we can find and with the best compiler options, in an attempt to make it suitable for a high-load server. For a typical user, even for setups with a lot of concurrent connections or tables exceeding the 2GB limit, our binary is the best choice in most cases. After reading the following text, if you are in doubt about what to do, try our binary first to determine whether it meets your needs. If you discover that it is not good enough, you may want to try your own build. In that case, we would appreciate a note about it so that we can build a better binary next time.

MySQL uses LinuxThreads on Linux. If you are using an old Linux version that does not have `glibc2`, you must install LinuxThreads before trying to compile MySQL. You can obtain LinuxThreads at http://dev.mysql.com/downloads/os-linux.html.

Note that `glibc` versions before and including version 2.1.1 have a fatal bug in `pthread_mutex_timedwait()` handling, which is used when you issue `INSERT DELAYED` statements. Do not use `INSERT DELAYED` before upgrading `glibc`.

Note that Linux kernel and the LinuxThread library can by default have only 1,024 threads. If you plan to have more than 1,000 concurrent connections, you need to make some changes to LinuxThreads:

- Increase `PTHREAD_THREADS_MAX` in `sysdeps/unix/sysv/linux/bits/local_lim.h` to 4096 and decrease `STACK_SIZE` in `linuxthreads/internals.h` to 256KB. The paths are relative to the root of `glibc`. (Note that MySQL is not stable with around 600 to 1000 connections if `STACK_SIZE` is the default of 2MB.)
- Recompile LinuxThreads to produce a new `libpthread.a` library, and relink MySQL against it.

There is another issue that greatly hurts MySQL performance, especially on SMP systems. The mutex implementation in LinuxThreads in `glibc` 2.1 is very bad for programs with many threads that hold the mutex only for a short time. This produces a paradoxical result: If you link MySQL against an unmodified LinuxThreads, removing processors from an SMP actually improves MySQL performance in many cases. We have made a patch available for `glibc` 2.1.3 to correct this behavior (http://dev.mysql.com/Downloads/Linux/linuxthreads-2.1-patch).

With `glibc` 2.2.2, MySQL 3.23.36 uses the adaptive mutex, which is much better than even the patched one in `glibc` 2.1.3. Be warned, however, that under some conditions, the current mutex code in `glibc` 2.2.2 overspins, which hurts MySQL performance. The likelihood that this condition occurs can be reduced by renicing the `mysqld` process to the highest priority. We have also been able to correct the overspin behavior with a patch, available at http://dev.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch. It combines the correction of overspin, maximum number of threads, and stack spacing all in one. You need to apply it in the `linuxthreads` directory with `patch -p0 </tmp/linuxthreads-2.2.2.patch`. We hope it is included in some form in future releases of `glibc` 2.2. In any case, if you link against `glibc` 2.2.2, you still need to correct `STACK_SIZE` and `PTHREAD_THREADS_MAX`. We hope that the defaults is corrected to some more acceptable values for high-load MySQL setup in the future, so that the commands needed to produce your own build can be reduced to `./configure; make; make install`.

If you use these patches to build a special static version of `libpthread.a`, use it only for statically linking against MySQL. We know that the patches are safe for MySQL and significantly improve its performance, but we cannot say anything about other applications. If you link other applications that require LinuxThreads against the patched static version of the library, or build a patched shared version and install it on your system, you do so at your own risk.

If you experience any strange problems during the installation of MySQL, or with some common utilities hanging, it is very likely that they are either library or compiler related. If this is the case, using our binary resolves them.

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqclient.so` to `/usr/lib`.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you are using the Fujitsu compiler (`fcc/FCC`), you may have some problems compiling MySQL because the Linux header files are very `gcc` oriented. The following `configure` line should work with `fcc/FCC`:

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE \
    -DCONST=const -DNO_STRTOLL_PROTO" \
CXX=FCC CXXFLAGS="-O -K fast -K lib \
    -K omitfp -K preex --no_exceptions --no_rtti -D_GNU_SOURCE \
    -DCONST=const -Dalloca=__builtin_alloca -DNO_STRTOLL_PROTO \
    '-D_EXTERN_INLINE=static __inline'" \
./configure \
    --prefix=/usr/local/mysql --enable-assembler \
    --with-mysqld-ldflags=-all-static --disable-shared \
    --with-low-memory
```

2.12.1.4 Linux Postinstallation Notes

`mysql.server` can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See Section 2.10.2.2, “Starting and Stopping MySQL Automatically”.

If MySQL cannot open enough files or connections, it may be that you have not configured Linux to handle enough files.

In Linux 2.2 and onward, you can check the number of allocated file handles as follows:

```
shell> cat /proc/sys/fs/file-max
shell> cat /proc/sys/fs/dquot-max
shell> cat /proc/sys/fs/super-max
```

If you have more than 16MB of memory, you should add something like the following to your init scripts (for example, `/etc/init.d/boot.local` on SuSE Linux):

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

You can also run the `echo` commands from the command line as `root`, but these settings are lost the next time your computer restarts.

Alternatively, you can set these parameters on startup by using the `sysctl` tool, which is used by many Linux distributions (SuSE has added it as well, beginning with SuSE Linux 8.0). Just put the following values into a file named `/etc/sysctl.conf`:

```
# Increase some values for MySQL
```

```
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

You should also add the following to `/etc/my.cnf`:

```
[mysqld_safe]
open-files-limit=8192
```

This should enable a server limit of 8,192 for the combined number of connections and open files.

The `STACK_SIZE` constant in LinuxThreads controls the spacing of thread stacks in the address space. It needs to be large enough so that there is plenty of room for each individual thread stack, but small enough to keep the stack of some threads from running into the global `mysqld` data. Unfortunately, as we have discovered, the Linux implementation of `mmap()` successfully unmaps a mapped region if you ask it to map out an address currently in use, zeroing out the data on the entire page instead of returning an error. So, the safety of `mysqld` or any other threaded application depends on "gentlemanly" behavior of the code that creates threads. The user must take measures to make sure that the number of running threads at any time is sufficiently low for thread stacks to stay away from the global heap. With `mysqld`, you should enforce this behavior by setting a reasonable value for the `max_connections` [419] variable.

If you build MySQL yourself, you can patch LinuxThreads for better stack use. See Section 2.12.1.3, "Linux Source Distribution Notes". If you do not want to patch LinuxThreads, you should set `max_connections` [419] to a value no higher than 500. It should be even less if you have a large key buffer, large heap tables, or some other things that make `mysqld` allocate a lot of memory, or if you are running a 2.2 kernel with a 2GB patch. If you are using our binary or RPM version 3.23.25 or later, you can safely set `max_connections` [419] at 1500, assuming no large key buffer or heap tables with lots of data. The more you reduce `STACK_SIZE` in LinuxThreads the more threads you can safely create. Values between 128KB and 256KB are recommended.

If you use a lot of concurrent connections, you may suffer from a "feature" in the 2.2 kernel that attempts to prevent fork bomb attacks by penalizing a process for forking or cloning a child. This causes MySQL not to scale well as you increase the number of concurrent clients. On single-CPU systems, we have seen this manifested as very slow thread creation: It may take a long time to connect to MySQL (as long as one minute), and it may take just as long to shut it down. On multiple-CPU systems, we have observed a gradual drop in query speed as the number of clients increases. In the process of trying to find a solution, we have received a kernel patch from one of our users who claimed it made a lot of difference for his site. The patch is available at http://dev.mysql.com/Downloads/Patches/linux-fork.patch. We have done rather extensive testing of this patch on both development and production systems. It has significantly improved MySQL performance without causing any problems and is recommended for users who still run high-load servers on 2.2 kernels.

This issue has been fixed in the 2.4 kernel, so if you are not satisfied with the current performance of your system, rather than patching your 2.2 kernel, it might be easier to upgrade to 2.4. On SMP systems, upgrading also gives you a nice SMP boost in addition to fixing the fairness bug.

We have tested MySQL on the 2.4 kernel on a two-CPU machine and found MySQL scales *much* better. There was virtually no slowdown on query throughput all the way up to 1,000 clients, and the MySQL scaling factor (computed as the ratio of maximum throughput to the throughput for one client) was 180%. We have observed similar results on a four-CPU system: Virtually no slowdown as the number of clients was increased up to 1,000, and a 300% scaling factor. Based on these results, for a high-load SMP server using a 2.2 kernel, it is definitely recommended to upgrade to the 2.4 kernel at this point.

We have discovered that it is essential to run the `mysqld` process with the highest possible priority on the 2.4 kernel to achieve maximum performance. This can be done by adding a `renice -20 $$` command to

`mysqld_safe`. In our testing on a four-CPU machine, increasing the priority resulted in a 60% throughput increase with 400 clients.

We are currently also trying to collect more information on how well MySQL performs with a 2.4 kernel on four-way and eight-way systems. If you have access such a system and have done some benchmarks, please send an email message to `<benchmarks@mysql.com>` with the results. We will review them for inclusion in the manual.

If you see a dead `mysqld` server process with `ps`, this usually means that you have found a bug in MySQL or that you have a corrupted table. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.

To get a core dump on Linux if `mysqld` dies with a `SIGSEGV` signal, you can start `mysqld` with the `--core-file` [385] option. Note that you also probably need to raise the core file size by adding `ulimit -c 1000000` to `mysqld_safe` or starting `mysqld_safe` with `--core-file-size=1000000` [244]. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

2.12.1.5 Linux x86 Notes

MySQL requires `libc` 5.4.12 or newer. It is known to work with `libc` 5.4.46. `glibc` 2.0.6 and later should also work. There have been some problems with the `glibc` RPMs from Red Hat, so if you have problems, check whether there are any updates. The `glibc` 2.0.7-19 and 2.0.7-29 RPMs are known to work.

If you are using Red Hat 8.0 or a new `glibc` 2.2.x library, you may see `mysqld` die in `gethostbyaddr()`. This happens because the new `glibc` library requires a stack size greater than 128KB for this call. To fix the problem, start `mysqld` with the `--thread-stack=192K` [431] option. This stack size is the default on MySQL 4.0.10 and above, so you should not see the problem.

If you are using `gcc` 3.0 and above to compile MySQL, you must install the `libstdc++v3` library before compiling MySQL; if you do not do this, you get an error about a missing `__cxa_pure_virtual` symbol during linking.

On some older Linux distributions, `configure` may produce an error like this:

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Just do what the error message says. Add an extra underscore to the `_P` macro name that has only one underscore, and then try again.

You may get some warnings when compiling. Those shown here can be ignored:

```
mysqld.cc -o objs-thread/mysqld.o
mysqld.cc: In function `void init_signals()':
mysqld.cc:315: warning: assignment of negative value `-1' to
`long unsigned int'
mysqld.cc: In function `void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value `-1' to
`long unsigned int'
```

If `mysqld` always dumps core when it starts, the problem may be that you have an old `/lib/libc.a`. Try renaming it, and then remove `sql/mysqld` and do a new `make install` and try again. This problem has been reported on some Slackware installations.

If you get the following error when linking `mysqld`, it means that your `libg++.a` is not installed correctly:

```
```

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

You can avoid using `libg++.a` by running `configure` like this:

```
shell> CXX=gcc ./configure
```

If `mysqld` crashes immediately and you are running Red Hat 5.0 with a version of `glibc` older than 2.0.7-5, you should make sure that you have installed all `glibc` patches. There is a lot of information about this in the MySQL mail archives, available online at http://lists.mysql.com/.

2.12.1.6 Linux SPARC Notes

In some implementations, `readdir_r()` is broken. The symptom is that the `SHOW DATABASES` statement always returns an empty set. This can be fixed by removing `HAVE_READDIR_R` from `config.h` after configuring and before compiling.

2.12.1.7 Linux Alpha Notes

MySQL 3.23.12 is the first MySQL version that is tested on Linux-Alpha. If you plan to use MySQL on Linux-Alpha, you should ensure that you have this version or newer.

We have tested MySQL on Alpha with our benchmarks and test suite, and it appears to work nicely.

We currently build the MySQL binary packages on SuSE Linux 7.0 for AXP, kernel 2.4.4-SMP, Compaq C compiler (V6.2-505) and Compaq C++ compiler (V6.3-006) on a Compaq DS20 machine with an Alpha EV6 processor.

You can find the preceding compilers at http://www.support.compaq.com/alpha-tools/. By using these compilers rather than `gcc`, we get about 9% to 14% better MySQL performance.

Note that until MySQL version 3.23.52 and 4.0.2, we optimized the binary for the current CPU only (by using the `-fast` compile option). This means that for older versions, you can use our Alpha binaries only if you have an Alpha EV6 processor.

For all subsequent releases, we added the `-arch generic` flag to our compile options, which ensures that the binary runs on all Alpha processors. We also compile statically to avoid library problems. The `configure` command looks like this:

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx \
CXXFLAGS="-fast -arch generic -noexceptions -nortti" \
./configure --prefix=/usr/local/mysql --disable-shared \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared
```

Some known problems when running MySQL on Linux-Alpha:

- Debugging threaded applications like MySQL does not work with `gdb 4.18`. You should use `gdb` 5.1 instead.
- If you try linking `mysqld` statically when using `gcc`, the resulting image dumps core at startup time. In other words, *do not* use `--with-mysqld-ldflags=-all-static` with `gcc`.

2.12.1.8 Linux PowerPC Notes

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc` 2.0.7).

2.12.1.9 Linux MIPS Notes

To get MySQL to work on Qube2 (Linux Mips), you need the newest `glibc` libraries. `glibc-2.0.7-29C2` is known to work. You must also use `gcc` 2.95.2 or newer).

2.12.1.10 Linux IA-64 Notes

To get MySQL to compile on Linux IA-64, we use the following `configure` command for building with `gcc` 2.96:

```
CC=gcc \
CFLAGS="-O3 -fno-omit-frame-pointer" \
CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
    ./configure --prefix=/usr/local/mysql \
    "--with-comment=Official MySQL binary" \
    --with-extra-charsets=complex
```

On IA-64, the MySQL client binaries use shared libraries. This means that if you install our binary distribution at a location other than `/usr/local/mysql`, you need to add the path of the directory where you have `libmysqlclient.so` installed either to the `/etc/ld.so.conf` file or to the value of your `LD_LIBRARY_PATH` environment variable.

See Section 17.6.3.1, “Building C API Client Programs”.

2.12.1.11 SELinux Notes

RHEL4 comes with SELinux, which supports tighter access control for processes. If SELinux is enabled (`SELINUX` in `/etc/selinux/config` is set to `enforcing`, `SELINUXTYPE` is set to either `targeted` or `strict`), you might encounter problems installing Oracle Corporation RPM packages.

Red Hat has an update that solves this. It involves an update of the “security policy” specification to handle the install structure of the RPMs provided by Oracle Corporation. For further information, see https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=167551 and http://rhn.redhat.com/errata/RHBA-2006-0049.html.

The preceding discussion applies only to RHEL4. The patch is unnecessary for RHEL5.

2.12.2 Mac OS X Notes

On Mac OS X, `tar` cannot handle long file names. If you need to unpack a `.tar.gz` distribution, use `gnutar` instead.

2.12.2.1 Mac OS X 10.x (Darwin)

MySQL should work without major problems on Mac OS X 10.x (Darwin).

Known issues:

- If you have problems with performance under heavy load, try using the `--skip-thread-priority` [394] option to `mysqld`. This runs all threads with the same priority. On Mac OS X, this gives better performance, at least until Apple fixes its thread scheduler.
- The connection times (`wait_timeout` [434], `interactive_timeout` [414] and `net_read_timeout` [422]) values are not honored. The symptom is that persistent connections can

hang for a very long time without getting closed down and that a 'kill' for a thread will not take affect until the thread does it a new command

This is probably a signal handling problem in the thread library where the signal does not break a pending read and we hope that a future update to the thread libraries will fix this.

Our binary for Mac OS X is compiled on Darwin 6.3 with the following `configure` line:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
    ./configure --prefix=/usr/local/mysql \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --enable-local-infile --disable-shared
```

See Section 2.5, “Installing MySQL on Mac OS X”.

2.12.2.2 Mac OS X Server 1.2 (Rhapsody)

For current versions of Mac OS X Server, no operating system changes are necessary before compiling MySQL. Compiling for the Server platform is the same as for the client version of Mac OS X. (However, note that MySQL comes preinstalled on Mac OS X Server, so you need not build it yourself.)

For older versions (Mac OS X Server 1.2, a.k.a. Rhapsody), you must first install a pthread package before trying to configure MySQL.

See Section 2.5, “Installing MySQL on Mac OS X”.

2.12.3 Solaris Notes

For information about installing MySQL on Solaris using PKG distributions, see Section 2.6, “Installing MySQL on Solaris”.

On Solaris, you may run into trouble even before you get the MySQL distribution unpacked. Solaris `tar` cannot handle long file names, so you may see an error like this when you unpack MySQL:

```
x mysql-3.22.12-beta/bench/Results/ATIS-mysql_odbc-NT_4.0-cmp-db2,
informix,ms-sql,mysql,oracle,solid,sybase, 0 bytes, 0 tape blocks
tar: directory checksum error
```

In this case, you must use GNU `tar` (`gtar`) to unpack the distribution.

Sun native threads work only on Solaris 2.5 and higher. For Solaris 2.4 and earlier, MySQL automatically uses MIT-pthreads. See Section 2.9.6, “MIT-pthreads Notes”.

If you get the following error from `configure`, it means that you have something wrong with your compiler installation:

```
checking for restartable system calls... configure: error cannot
run test programs while cross compiling
```

In this case, you should upgrade your compiler to a newer version. You may also be able to solve this problem by inserting the following row into the `config.cache` file:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

If you are using Solaris on a SPARC, the recommended compiler is `gcc` 2.95.2 or 3.2. You can find this at http://gcc.gnu.org/. Note that `gcc` 2.8.1 does not work reliably on SPARC.

The recommended `configure` line when using `gcc` 2.95.2 is:

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory \
    --enable-assembler
```

If you have an UltraSPARC system, you can get 4% better performance by adding `-mcpu=v8 -Wa,-xarch=v8plusa` to the `CFLAGS` and `CXXFLAGS` environment variables.

If you have Sun's Forte 5.0 (or newer) compiler, you can run `configure` like this:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit binary with Sun's Forte compiler, use the following configuration options:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit Solaris binary using `gcc`, add `-m64` to `CFLAGS` and `CXXFLAGS` and remove `--enable-assembler` from the `configure` line. This works only with MySQL 4.0 and up; MySQL 3.23 does not include the required modifications to support this.

In the MySQL benchmarks, we got a 4% speedup on an UltraSPARC when using Forte 5.0 in 32-bit mode compared to using `gcc` 3.2 with the `-mcpu` flag.

If you create a 64-bit `mysqld` binary, it is 4% slower than the 32-bit binary, but can handle more threads and memory.

When using Solaris 10 for x86_64, you should mount any file systems on which you intend to store `InnoDB` files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so will cause a significant drop in performance when using the `InnoDB` storage engine on this platform.

If you get a problem with `fdatasync` or `sched_yield`, you can fix this by adding `LIBS=-lrt` to the `configure` line

For compilers older than WorkShop 5.3, you might have to edit the `configure` script. Change this line:

```
#if !defined(__STDC__) || __STDC__ != 1
```

To this:

```
#if !defined(__STDC__)
```

If you turn on `__STDC__` with the `-Xc` option, the Sun compiler cannot compile with the Solaris `pthread.h` header file. This is a Sun bug (broken compiler or broken include file).

If `mysqld` issues the following error message when you run it, you have tried to compile MySQL with the Sun compiler without enabling the `-mt` multi-thread option:

```
libc internal error: _rmutex_unlock: rmutex not held
```

Add `-mt` to `CFLAGS` and `CXXFLAGS` and recompile.

If you are using the SFW version of `gcc` (which comes with Solaris 8), you must add `/opt/sfw/lib` to the environment variable `LD_LIBRARY_PATH` before running `configure`.

If you are using the `gcc` available from `sunfreeware.com`, you may have many problems. To avoid this, you should recompile `gcc` and GNU `binutils` on the machine where you are running them.

If you get the following error when compiling MySQL with `gcc`, it means that your `gcc` is not configured for your version of Solaris:

```
shell> gcc -O3 -g -O2 -DDBUG_OFF  -o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

The proper thing to do in this case is to get the newest version of `gcc` and compile it with your current `gcc` compiler. At least for Solaris 2.5, almost all binary versions of `gcc` have old, unusable include files that break all programs that use threads, and possibly other programs!

Solaris does not provide static versions of all system libraries (`libpthreads` and `libdl`), so you cannot compile MySQL with `--static`. If you try to do so, you get one of the following errors:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r`*`/full/path/to/`*`libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqclient.so` to `/usr/lib`.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you have problems with `configure` trying to link with `-lz` when you do not have `zlib` installed, you have two options:

- If you want to be able to use the compressed communication protocol, you need to get and install `zlib` from `ftp.gnu.org`.
- Run `configure` with the `--with-named-z-libs=no` option when building MySQL.

If you are using `gcc` and have problems with loading user-defined functions (UDFs) into MySQL, try adding `-lgcc` to the link line for the UDF.

If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.

If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

You might try starting the server with the `--back_log=50` [406] option as a workaround for this. (Use `-O back_log=50` before MySQL 4.)

Solaris does not support core files for `setuid()` applications, so you cannot get a core file from `mysqld` if you are using the `--user` [395] option.

2.12.3.1 Solaris 2.7/2.8 Notes

Normally, you can use a Solaris 2.6 binary on Solaris 2.7 and 2.8. Most of the Solaris 2.6 issues also apply for Solaris 2.7 and 2.8.

MySQL 3.23.4 and above should be able to detect new versions of Solaris automatically and enable workarounds for the following problems.

Solaris 2.7 and 2.8 have some bugs in the include files. You may see the following error when you use `gcc`:

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

If this occurs, you can fix the problem by copying `/usr/include/widec.h` to `.../lib/gcc-lib/os/gcc-version/include` and changing line 41 from this:

```
#if     !defined(lint) && !defined(__lint)
```

To this:

```
#if     !defined(lint) && !defined(__lint) && !defined(getwc)
```

Alternatively, you can edit `/usr/include/widec.h` directly. Either way, after you make the fix, you should remove `config.cache` and run `configure` again.

If you get the following errors when you run `make`, it is because `configure` did not detect the `curses.h` file (probably because of the error in `/usr/include/widec.h`):

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

The solution to this problem is to do one of the following:

1. Configure with `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure`.
2. Edit `/usr/include/widec.h` as indicated in the preceding discussion and re-run `configure`.
3. Remove the `#define HAVE_TERM` line from the `config.h` file and run `make` again.

If your linker cannot find `-lz` when linking client programs, the problem is probably that your `libz.so` file is installed in `/usr/local/lib`. You can fix this problem by one of the following methods:

- Add `/usr/local/lib` to `LD_LIBRARY_PATH`.

- Add a link to `libz.so` from `/lib`.
- If you are using Solaris 8, you can install the optional `zlib` from your Solaris 8 CD distribution.
- Run `configure` with the `--with-named-z-libs=no` option when building MySQL.

2.12.3.2 Solaris x86 Notes

On Solaris 8 on x86, `mysqld` dumps core if you remove the debug symbols using `strip`.

If you are using `gcc` on Solaris x86 and you experience problems with core dumps under load, you should use the following `configure` command:

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \
CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti -DHAVE_CURSES_H" \
./configure --prefix=/usr/local/mysql
```

This avoids problems with the `libstdc++` library and with C++ exceptions.

If this does not help, you should compile a debug version and run it with a trace file or under `gdb`. See Section 18.4, “Porting to Other Systems”.

2.12.4 BSD Notes

This section provides information about using MySQL on variants of BSD Unix.

2.12.4.1 FreeBSD Notes

FreeBSD 4.x or newer is recommended for running MySQL, because the thread package is much more integrated. To get a secure and stable system, you should use only FreeBSD kernels that are marked `-RELEASE`.

The easiest (and preferred) way to install MySQL is to use the `mysql-server` and `mysql-client` ports available at http://www.freebsd.org/. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

It is recommended you use MIT-pthreads on FreeBSD 2.x, and native threads on FreeBSD 3 and up. It is possible to run with native threads on some late 2.2.x versions, but you may encounter problems shutting down `mysqld`.

Unfortunately, certain function calls on FreeBSD are not yet fully thread-safe. Most notably, this includes the `gethostbyname()` function, which is used by MySQL to convert host names into IP addresses. Under certain circumstances, the `mysqld` process suddenly causes 100% CPU load and is unresponsive. If you encounter this problem, try to start MySQL using the `--skip-name-resolve` [393] option.

Alternatively, you can link MySQL on FreeBSD 4.x against the LinuxThreads library, which avoids a few of the problems that the native FreeBSD thread implementation has. For a very good comparison of

LinuxThreads versus native threads, see Jeremy Zawodny's article *FreeBSD or Linux for your MySQL Server?* at http://jeremy.zawodny.com/blog/archives/000697.html.

Known problem when using LinuxThreads on FreeBSD is:

- The connection times (`wait_timeout` [434], `interactive_timeout` [414] and `net_read_timeout` [422]) values are not honored. The symptom is that persistent connections can hang for a very long time without getting closed down and that a 'kill' for a thread will not take affect until the thread does it a new command

  This is probably a signal handling problem in the thread library where the signal does not break a pending read. This is supposed to be fixed in FreeBSD 5.0

The MySQL build process requires GNU make (`gmake`) to work. If GNU `make` is not available, you must install it first before compiling MySQL.

The recommended way to compile and install MySQL on FreeBSD with `gcc` (2.95.2 and up) is:

```
CC=gcc CFLAGS="-O2 -fno-strength-reduce" \
    CXX=gcc CXXFLAGS="-O2 -fno-rtti -fno-exceptions \
    -felide-constructors -fno-strength-reduce" \
    ./configure --prefix=/usr/local/mysql --enable-assembler
gmake
gmake install
cd /usr/local/mysql
bin/mysql_install_db --user=mysql
bin/mysqld_safe &
```

If you notice that `configure` uses MIT-pthreads, you should read the MIT-pthreads notes. See Section 2.9.6, “MIT-pthreads Notes”.

If you get an error from `make install` that it cannot find `/usr/include/pthreads`, `configure` did not detect that you need MIT-pthreads. To fix this problem, remove `config.cache`, and then re-run `configure` with the `--with-mit-threads` option.

Be sure that your name resolver setup is correct. Otherwise, you may experience resolver delays or failures when connecting to `mysqld`. Also make sure that the `localhost` entry in the `/etc/hosts` file is correct. The file should start with a line similar to this:

```
127.0.0.1       localhost localhost.your.domain
```

FreeBSD is known to have a very low default file handle limit. See Section B.5.2.18, “`'File' Not Found` and Similar Errors”. Start the server by using the `--open-files-limit` [245] option for `mysqld_safe`, or raise the limits for the `mysqld` user in `/etc/login.conf` and rebuild it with `cap_mkdb /etc/login.conf`. Also be sure that you set the appropriate class for this user in the password file if you are not using the default (use `chpass mysqld-user-name`). See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

FreeBSD limits the size of a process to 512MB, even if you have much more RAM available on the system. So you may get an error such as this:

```
Out of memory (Needed 16391 bytes)
```

In current versions of FreeBSD (at least 4.x and greater), you may increase this limit by adding the following entries to the `/boot/loader.conf` file and rebooting the machine (these are not settings that can be changed at run time with the `sysctl` command):

```
kern.maxdsiz="1073741824" # 1GB
kern.dfldsiz="1073741824" # 1GB
kern.maxssiz="134217728" # 128MB
```

For older versions of FreeBSD, you must recompile your kernel to change the maximum data segment size for a process. In this case, you should look at the `MAXDSIZ` option in the `LINT` config file for more information.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Section 2.13, “Environment Variables”.

2.12.4.2 NetBSD Notes

To compile on NetBSD, you need GNU `make`. Otherwise, the build process fails when `make` tries to run `lint` on C++ files.

2.12.4.3 OpenBSD 2.5 Notes

On OpenBSD 2.5, you can compile MySQL with native threads with the following options:

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.12.4.4 BSD/OS Version 2.x Notes

If you get the following error when compiling MySQL, your `ulimit` value for virtual memory is too low:

```
item_func.h: In method
`Item_func_ge::Item_func_ge(const Item_func_ge &)':
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Try using `ulimit -v 80000` and run `make` again. If this does not work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

If you are using `gcc`, you may also use have to use the `--with-low-memory` [99] flag for `configure` to be able to compile `sql_yacc.cc`.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Section 2.13, “Environment Variables”.

2.12.4.5 BSD/OS Version 3.x Notes

Upgrade to BSD/OS 3.1. If that is not possible, install BSDIpatch M300-038.

Use the following command when configuring MySQL:

```
env CXX=shlicc++ CC=shlicc2 \
./configure \
    --prefix=/usr/local/mysql \
    --localstatedir=/var/mysql \
    --without-perl \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

The following is also known to work:

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \
```

```
./configure \
    --prefix=/usr/local/mysql \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

You can change the directory locations if you wish, or just use the defaults by not specifying any locations.

If you have problems with performance under heavy load, try using the `--skip-thread-priority` [394] option to `mysqld`. This runs all threads with the same priority. On BSDI 3.1, this gives better performance, at least until BSDI fixes its thread scheduler.

If you get the error `virtual memory exhausted` while compiling, you should try using `ulimit -v 80000` and running `make` again. If this does not work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

2.12.4.6 BSD/OS Version 4.x Notes

BSDI 4.x has some thread-related bugs. If you want to use MySQL on this, you should install all thread-related patches. At least M400-023 should be installed.

On some BSDI 4.x systems, you may get problems with shared libraries. The symptom is that you cannot execute any client programs, for example, `mysqladmin`. In this case, you need to reconfigure not to use shared libraries with the `--disable-shared` option to configure.

Some customers have had problems on BSDI 4.0.1 that the `mysqld` binary after a while cannot open tables. This occurs because some library/system-related bug causes `mysqld` to change current directory without having asked for that to happen.

The fix is to either upgrade MySQL to at least version 3.23.34 or, after running `configure`, remove the line `#define HAVE_REALPATH` from `config.h` before running `make`.

Note that this means that you cannot symbolically link a database directories to another database directory or symbolic link a table to another database on BSDI. (Making a symbolic link to another disk is okay).

2.12.5 Other Unix Notes

2.12.5.1 HP-UX Version 10.20 Notes

If you install MySQL using a binary tarball distribution on HP-UX, you may run into trouble even before you get the MySQL distribution unpacked, as the HP-UX `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

There are a couple of small problems when compiling MySQL on HP-UX. Use `gcc` instead of the HP-UX native compiler, because `gcc` produces better code.

Use `gcc` 2.95 on HP-UX. Do not use high optimization flags (such as `-O6`) because they may not be safe on HP-UX.

The following `configure` line should work with `gcc` 2.95:

```
CFLAGS="-I/opt/dce/include -fpic" \
CXXFLAGS="-I/opt/dce/include -felide-constructors -fno-exceptions \
-fno-rtti" \
CXX=gcc \
./configure --with-pthread \
    --with-named-thread-libs='-ldce' \
    --prefix=/usr/local/mysql --disable-shared
```

The following `configure` line should work with `gcc` 3.1:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc \
CXXFLAGS="-DHPUX -I/opt/dce/include -felide-constructors \
    -fno-exceptions -fno-rtti -O3 -fPIC" \
./configure --prefix=/usr/local/mysql \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --enable-local-infile  --with-pthread \
    --with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC
    --disable-shared
```

2.12.5.2 HP-UX Version 11.x Notes

For HP-UX 11.x, use MySQL 3.23.15 or later.

If you install MySQL using a binary tarball distribution on HP-UX, you may run into trouble even before you get the MySQL distribution unpacked, as the HP-UX `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU `tar` (`gtar`) to unpack the distribution.

Because of some critical bugs in the standard HP-UX libraries, you should install the following patches before trying to run MySQL on HP-UX 11.0:

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

This solves the problem of getting `EWOULDBLOCK` from `recv()` and `EBADF` from `accept()` in threaded applications.

If you are using `gcc` 2.95.1 on an unpatched HP-UX 11.x system, you may get the following error:

```
In file included from /usr/include/unistd.h:11,
                 from ../include/global.h:125,
                 from mysql_priv.h:15,
                 from item.cc:19:
/usr/include/sys/unistd.h:184: declaration of C function ...
/usr/include/sys/pthread.h:440: previous declaration ...
In file included from item.h:306,
                 from mysql_priv.h:158,
                 from item.cc:19:
```

The problem is that HP-UX does not define `pthreads_atfork()` consistently. It has conflicting prototypes in `/usr/include/sys/unistd.h`:184 and `/usr/include/sys/pthread.h`:440.

One solution is to copy `/usr/include/sys/unistd.h` into `mysql/include` and edit `unistd.h` and change it to match the definition in `pthread.h`. Look for this line:

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
                                          void (*child)());
```

Change it to look like this:

```
extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),
                                          void (*child)(void));
```

After making the change, the following `configure` line should work:

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

If you are using MySQL 4.0.5 with the HP-UX compiler, you can use the following command (which has been tested with `cc` B.11.11.04):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
    --with-extra-character-set=complex
```

You can ignore any errors of the following type:

```
aCC: warning 901: unknown option: `-3': use +help for online
documentation
```

If you get the following error from `configure`, verify that you do not have the path to the K&R compiler before the path to the HP-UX C and C++ compiler:

```
checking for cc option to accept ANSI C... no
configure: error: MySQL requires an ANSI C compiler (and a C++ compiler).
Try gcc. See the Installation chapter in the Reference Manual.
```

Another reason for not being able to compile is that you did not define the `+DD64` flags as just described.

Another possibility for HP-UX 11 is to use MySQL binaries for HP-UX 10.20. We have received reports from some users that these binaries work fine on HP-UX 11.00. If you encounter problems, be sure to check your HP-UX patch level.

2.12.5.3 IBM-AIX notes

Automatic detection of `xlC` is missing from Autoconf, so a number of variables need to be set before running `configure`. The following example uses the IBM compiler:

```
export CC="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192 "
export CXX="xlC_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CFLAGS="-I /usr/local/include"
export LDFLAGS="-L /usr/local/lib"
export CPPFLAGS=$CFLAGS
export CXXFLAGS=$CFLAGS

./configure --prefix=/usr/local \
                --localstatedir=/var/mysql \
                --sbindir='/usr/local/bin' \
                --libexecdir='/usr/local/bin' \
                --enable-thread-safe-client \
                --enable-large-files
```

The preceding options are used to compile the MySQL distribution that can be found at http://www-frec.bull.com/.

If you change the `-O3` to `-O2` in the preceding `configure` line, you must also remove the `-qstrict` option. This is a limitation in the IBM C compiler.

If you are using `gcc` to compile MySQL, you *must* use the `-fno-exceptions` flag, because the exception handling in `gcc` is not thread-safe! There are also some known problems with IBM's assembler that may cause it to generate bad code when used with `gcc`.

Use the following `configure` line with `gcc` 2.95 on AIX:

```
CC="gcc -pipe -mcpu=power -Wa,-many" \
CXX="gcc -pipe -mcpu=power -Wa,-many" \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory
```

The `-Wa,-many` option is necessary for the compile to be successful. IBM is aware of this problem but is in no hurry to fix it because of the workaround that is available. We do not know if the `-fno-exceptions` is required with `gcc` 2.95, but because MySQL does not use exceptions and the option generates faster code, you should always use it with `gcc`.

If you get a problem with assembler code, try changing the `-mcpu=xxx` option to match your CPU. Typically `power2`, `power`, or `powerpc` may need to be used. Alternatively, you might need to use `604` or `604e`. We are not positive but suspect that `power` would likely be safe most of the time, even on a power2 machine.

If you do not know which CPU is present, execute a `uname -m` command. It produces a string that looks like `000514676700` whose format is `xxyyyyyymmss` where `xx` and `ss` are always `00`, `yyyyyy` is a unique system ID and `mm` is the ID of the CPU Planar. A chart of these values can be found at http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm.

This gives you a machine type and model which you can use to determine what type of CPU you have.

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring as follows:

```
CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti \
-DDONT_USE_THR_ALARM" \
./configure --prefix=/usr/local/mysql --with-debug \
    --with-low-memory
```

This does not affect the performance of MySQL, but has the side effect that you cannot kill clients that are “sleeping” on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client dies when it issues its next command.

On some versions of AIX, linking with `libbind.a` makes `getservbyname()` dump core. This is an AIX bug and should be reported to IBM.

For AIX 4.2.1 and `gcc`, you have to make the following changes.

After configuring, edit `config.h` and `include/my_config.h` and change the line that says this:

```
#define HAVE_SNPRINTF 1
```

to this:

```
#undef HAVE_SNPRINTF
```

And finally, in `mysqld.cc`, you need to add a prototype for `initgroups()`.

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

For 32-bit binaries, if you need to allocate a lot of memory to the `mysqld` process, it is not sufficient merely to use `ulimit -d unlimited`. You may also have to modify `mysqld_safe`, adding a line something like this:

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

You can find more information about using very large amounts of memory at http://publib16.boulder.ibm.com/pseries/en_US/aixprggd/genprogc/lrg_prg_support.htm.

Users of AIX 4.3 should use `gmake` instead of the `make` utility included with AIX.

2.12.5.4 SunOS 4 Notes

On SunOS 4, MIT-pthreads is needed to compile MySQL. This in turn means you need GNU `make`.

Some SunOS 4 systems have problems with dynamic libraries and `libtool`. You can use the following `configure` line to avoid this problem:

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling `readline`, you may get warnings about duplicate defines. These can be ignored.

When compiling `mysqld`, there are some `implicit declaration of function` warnings. These can be ignored.

2.12.5.5 Alpha-DEC-UNIX Notes (Tru64)

If you are using `egcs` 1.1.2 on Digital Unix, you should upgrade to `gcc` 2.95.2, because `egcs` on DEC has some serious bugs.

When compiling threaded programs under Digital Unix, the documentation recommends using the `-pthread` option for `cc` and `cxx` and the `-lmach -lexc` libraries (in addition to `-lpthread`). You should run `configure` something like this:

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

When compiling `mysqld`, you may see a couple of warnings like this:

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

You can safely ignore these warnings. They occur because `configure` can detect only errors, not warnings.

If you start the server directly from the command line, you may have problems with it dying when you log out. (When you log out, your outstanding processes receive a `SIGHUP` signal.) If so, try starting the server like this:

```
nohup mysqld [options] &
```

`nohup` causes the command following it to ignore any `SIGHUP` signal sent from the terminal. Alternatively, start the server by running `mysqld_safe`, which invokes `mysqld` using `nohup` for you. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

If you get a problem when compiling `mysys/get_opt.c`, just remove the `#define _NO_PROTO` line from the start of that file.

If you are using Compaq's CC compiler, the following `configure` line should work:

```
CC="cc -pthread"
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
        -speculate all -arch host"
CXX="cxx -pthread"
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
          -speculate all -arch host -noexceptions -nortti"
export CC CFLAGS CXX CXXFLAGS
./configure \
    --prefix=/usr/local/mysql \
    --with-low-memory \
    --enable-large-files \
    --enable-shared=yes \
    --with-named-thread-libs="-lpthread -lmach -lexc -lc"
gnumake
```

If you get a problem with `libtool` when compiling with shared libraries as just shown, when linking `mysql`, you should be able to get around this by issuing these commands:

```
cd mysql
/bin/sh ../libtool --mode=link cxx -pthread  -O3 -DDBUG_OFF \
    -O4 -ansi_alias -ansi_args -fast -inline speed \
    -speculate all \ -arch host  -DUNDEF_HAVE_GETHOSTBYNAME_R \
    -o mysql  mysql.o readline.o sql_string.o completion_hash.o \
    ../readline/libreadline.a -lcurses \
    ../libmysql/.libs/libmysqlclient.so  -lm
cd ..
gnumake
gnumake install
scripts/mysql_install_db
```

2.12.5.6 Alpha-DEC-OSF/1 Notes

If you have problems compiling and have DEC `CC` and `gcc` installed, try running `configure` like this:

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

If you get problems with the `c_asm.h` file, you can create and use a 'dummy' `c_asm.h` file with:

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

Note that the following problems with the `ld` program can be fixed by downloading the latest DEC (Compaq) patch kit from: http://ftp.support.compaq.com/public/unix/.

On OSF/1 V4.0D and compiler "DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)," the compiler had some strange behavior (undefined `asm` symbols). `/bin/ld` also appears to be broken (problems with `_exit undefined` errors occurring while linking `mysqld`). On this system, we have managed to compile MySQL with the following `configure` line, after replacing `/bin/ld` with the version from OSF 4.0C:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

With the Digital compiler "C++ V6.1-029," the following should work:

```
CC=cc -pthread
CFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
       -speculate all -arch host
CXX=cxx -pthread
CXXFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
         -speculate all -arch host -noexceptions -nortti
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql \
            --with-mysqld-ldflags=-all-static --disable-shared \
            --with-named-thread-libs="-lmach -lexc -lc"
```

In some versions of OSF/1, the `alloca()` function is broken. Fix this by removing the line in `config.h` that defines `'HAVE_ALLOCA'`.

The `alloca()` function also may have an incorrect prototype in `/usr/include/alloca.h`. This warning resulting from this can be ignored.

`configure` uses the following thread libraries automatically: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

When using `gcc`, you can also try running `configure` like this:

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring with:

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

This does not affect the performance of MySQL, but has the side effect that you cannot kill clients that are “sleeping” on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client dies when it issues its next command.

With `gcc` 2.95.2, you may encounter the following compile error:

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

To fix this, you should change to the `sql` directory and do a cut-and-paste of the last `gcc` line, but change `-O3` to `-O0` (or add `-O0` immediately after `gcc` if you do not have any `-O` option on your compile line). After this is done, you can just change back to the top-level directory and run `make` again.

2.12.5.7 SGI Irix Notes

If you are using Irix 6.5.3 or newer, `mysqld` is able to create threads only if you run it as a user that has `CAP_SCHED_MGT` privileges (such as `root`) or if you give the `mysqld` server this privilege with the following shell command:

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

You may have to undefine some symbols in `config.h` after running `configure` and before compiling.

In some Irix implementations, the `alloca()` function is broken. If the `mysqld` server dies on some `SELECT` statements, remove the lines from `config.h` that define `HAVE_ALLOC` and `HAVE_ALLOCA_H`. If `mysqladmin create` does not work, remove the line from `config.h` that defines `HAVE_READDIR_R`. You may have to remove the `HAVE_TERM_H` line as well.

SGI recommends that you install all the patches on this page as a set:

http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

At the very minimum, you should install the latest kernel rollup, the latest `rld` rollup, and the latest `libc` rollup.

You definitely need all the POSIX patches on this page, for pthreads support:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

If you get the something like the following error when compiling `mysql.cc`:

```
"/usr/include/curses.h", line 82: error(1084):
invalid combination of type
```

Type the following in the top-level directory of your MySQL source tree:

```
extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
make
```

There have also been reports of scheduling problems. If only one thread is running, performance is slow. Avoid this by starting another client. This may lead to a two-to-tenfold increase in execution speed thereafter for the other thread. This is a poorly understood problem with Irix threads; you may have to improvise to find solutions until this can be fixed.

If you are compiling with `gcc`, you can use the following `configure` command:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
    --with-named-thread-libs=-lpthread
```

On Irix 6.5.11 with native Irix C and C++ compilers ver. 7.3.1.2, the following is reported to work

```
CC=cc CXX=CC CFLAGS='-O3 -n32 -TARG:platform=IP22 -I/usr/local/include \
-L/usr/local/lib' CXXFLAGS='-O3 -n32 -TARG:platform=IP22 \
-I/usr/local/include -L/usr/local/lib' \
./configure --prefix=/usr/local/mysql --with-innodb --with-berkeley-db \
    --with-libwrap=/usr/local \
    --with-named-curses-libs=/usr/local/lib/libncurses.a
```

2.12.5.8 SCO UNIX and OpenServer 5.0.x Notes

The current port is tested only on `sco3.2v5.0.5`, `sco3.2v5.0.6`, and `sco3.2v5.0.7` systems. There has also been progress on a port to `sco3.2v4.2`. Open Server 5.0.8 (Legend) has native threads and permits files greater than 2GB. The current maximum file size is 2GB.

We have been able to compile MySQL with the following `configure` command on OpenServer with `gcc` 2.95.3.

```
CC=gcc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
```

```
CXX=gcc CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-innodb \
    --with-openssl --with-vio --with-extra-charsets=complex
```

`gcc` is available at ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj.

This development system requires the OpenServer Execution Environment Supplement oss646B on OpenServer 5.0.6 and oss656B and the OpenSource libraries found in gwxlibs. All OpenSource tools are in the `opensrc` directory. They are available at ftp://ftp.sco.com/pub/openserver5/opensrc/.

Use the latest production release of MySQL.

SCO provides operating system patches at ftp://ftp.sco.com/pub/openserver5 for OpenServer 5.0.[0-6] and ftp://ftp.sco.com/pub/openserverv5/507 for OpenServer 5.0.7.

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenServer for OpenServer 5.0.x.

The maximum file size on an OpenServer 5.0.x system is 2GB.

The total memory which can be allocated for streams buffers, clists, and lock records cannot exceed 60MB on OpenServer 5.0.x.

Streams buffers are allocated in units of 4096 byte pages, clists are 70 bytes each, and lock records are 64 bytes each, so:

```
(NSTRPAGES * 4096) + (NCLIST * 70) + (MAX_FLCKREC * 64) <= 62914560
```

Follow this procedure to configure the Database Services option. If you are unsure whether an application requires this, see the documentation provided with the application.

1. Log in as `root`.
2. Enable the SUDS driver by editing the `/etc/conf/sdevice.d/suds` file. Change the `N` in the second field to a `Y`.
3. Use `mkdev aio` or the Hardware/Kernel Manager to enable support for asynchronous I/O and relink the kernel. To enable users to lock down memory for use with this type of I/O, update the aiomemlock(F) file. This file should be updated to include the names of users that can use AIO and the maximum amounts of memory they can lock down.
4. Many applications use setuid binaries so that you need to specify only a single user. See the documentation provided with the application to determine whether this is the case for your application.

After you complete this process, reboot the system to create a new kernel incorporating these changes.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

```
Value           Default         Min             Max
-----           -------         ---             ---
NBUF            0               24              450000
NHBUF           0               32              524288
NMPBUF          0               12              512
MAX_INODE       0               100             64000
MAX_FILE        0               100             64000
CTBUFSIZE       128             0               256
MAX_PROC        0               50              16000
```

```
MAX_REGION      0               500             160000
NCLIST          170             120             16640
MAXUP           100             15              16000
NOFILES         110             60              11000
NHINODE         128             64              8192
NAUTOUP         10              0               60
NGROUPS         8               0               128
BDFLUSHR        30              1               300
MAX_FLCKREC     0               50              16000
PUTBUFSZ        8000            2000            20000
MAXSLICE        100             25              100
ULIMIT          4194303         2048            4194303
* Streams Parameters
NSTREAM         64              1               32768
NSTRPUSH        9               9               9
NMUXLINK        192             1               4096
STRMSGSZ        16384           4096            524288
STRCTLSZ        1024            1024            1024
STRMAXBLK       524288          4096            524288
NSTRPAGES       500             0               8000
STRSPLITFRAC    80              50              100
NLOG            3               3               3
NUMSP           64              1               256
NUMTIM          16              1               8192
NUMTRW          16              1               8192
* Semaphore Parameters
SEMMAP          10              10              8192
SEMMNI          10              10              8192
SEMMNS          60              60              8192
SEMMNU          30              10              8192
SEMMSL          25              25              150
SEMOPM          10              10              1024
SEMUME          10              10              25
SEMVMX          32767           32767           32767
SEMAEM          16384           16384           16384
* Shared Memory Parameters
SHMMAX          524288          131072          2147483647
SHMMIN          1               1               1
SHMMNI          100             100             2000
FILE            0               100             64000
NMOUNT          0               4               256
NPROC           0               50              16000
NREGION         0               500             160000
```

Set these values as follows:

- `NOFILES` should be 4096 or 2048.
- `MAXUP` should be 2048.

To make changes to the kernel, use the `idtune` *`name parameter`* command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. For example, to change `SEMMS` to `200`, execute this command as `root`:

```
# /etc/conf/bin/idtune SEMMNS 200
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

To tune the system, the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- SHMMAX (recommended setting: 128MB) and SHMSEG (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- NOFILES and MAXUP should be set to at least 2048.
- MAXPROC should be set to at least 3000/4000 (depends on number of users) or more.
- The following formulas are recommended to calculate values for SEMMSL, SEMMNS, and SEMMNU:

```
SEMMSL = 13
```

13 is what has been found to be the best for both Progress and MySQL.

```
SEMMNS = SEMMSL * number of db servers to be run on the system
```

Set SEMMNS to the value of SEMMSL multiplied by the number of database servers (maximum) that you are running on the system at one time.

```
SEMMNU = SEMMNS
```

Set the value of SEMMNU to equal the value of SEMMNS. You could probably set this to 75% of SEMMNS, but this is a conservative estimate.

You need to at least install the SCO OpenServer Linker and Application Development Libraries or the OpenServer Development System to use gcc. You cannot use the GCC Dev system without installing one of these.

You should get the FSU Pthreads package and install it first. This can be found at http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz. You can also get a precompiled package from ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz.

FSU Pthreads can be compiled with SCO Unix 4.2 with tcpip, or using OpenServer 3.0 or Open Desktop 3.0 (OS 3.0 ODT 3.0) with the SCO Development System installed using a good port of GCC 2.5.x. For ODT or OS 3.0, you need a good port of GCC 2.5.x. There are a lot of problems without a good port. The port for this product requires the SCO Unix Development system. Without it, you are missing the libraries and the linker that is needed. You also need SCO-3.2v4.2-includes.tar.gz. This file contains the changes to the SCO Development include files that are needed to get MySQL to build. You need to replace the existing system include files with these modified header files. They can be obtained from ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

To build FSU Pthreads on your system, all you should need to do is run GNU make. The Makefile in FSU-threads-3.14.tar.gz is set up to make FSU-threads.

You can run ./configure in the threads/src directory and select the SCO OpenServer option. This command copies Makefile.SCO5 to Makefile. Then run make.

To install in the default /usr/include directory, log in as root, and then cd to the thread/src directory and run make install.

Remember that you must use GNU make to build MySQL.

Note

If you do not start mysqld_safe as root, you should get only the default 110 open files per process. mysqld writes a note about this in the log file.

With SCO 3.2V4.2, you should use FSU Pthreads version 3.14 or newer. The following `configure` command should work:

```
CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \
./configure \
    --prefix=/usr/local/mysql \
    --with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \
    --with-named-curses-libs="-lcurses"
```

You may have problems with some include files. In this case, you can find new SCO-specific include files at ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

You should unpack this file in the `include` directory of your MySQL source tree.

SCO development notes:

- MySQL should automatically detect FSU Pthreads and link `mysqld` with `-lgthreads -lsocket -lgthreads`.
- The SCO development libraries are re-entrant in FSU Pthreads. SCO claims that its library functions are re-entrant, so they must be re-entrant with FSU Pthreads. FSU Pthreads on OpenServer tries to use the SCO scheme to make re-entrant libraries.
- FSU Pthreads (at least the version at ftp://ftp.zenez.com) comes linked with GNU `malloc`. If you encounter problems with memory usage, make sure that `gmalloc.o` is included in `libgthreads.a` and `libgthreads.so`.
- In FSU Pthreads, the following system calls are pthreads-aware: `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()`, and `wait()`.
- The CSSA-2001-SCO.35.2 (the patch is listed in custom as erg711905-dscr_remap security patch (version 2.0.0)) breaks FSU threads and makes `mysqld` unstable. You have to remove this one if you want to run `mysqld` on an OpenServer 5.0.6 machine.
- If you use SCO OpenServer 5, you may need to recompile FSU pthreads with `-DDRAFT7` in `CFLAGS`. Otherwise, `InnoDB` may hang at a `mysqld` startup.
- SCO provides operating system patches at ftp://ftp.sco.com/pub/openserver5 for OpenServer 5.0.x.
- SCO provides security fixes and `libsocket.so.2` at ftp://ftp.sco.com/pub/security/OpenServer and ftp://ftp.sco.com/pub/security/sse for OpenServer 5.0.x.
- Pre-OSR506 security fixes. Also, the `telnetd` fix at ftp://stage.caldera.com/pub/security/openserver/ or ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/ as both `libsocket.so.2` and `libresolv.so.1` with instructions for installing on pre-OSR506 systems.

  It is probably a good idea to install these patches before trying to compile/use MySQL.

Beginning with Legend/OpenServer 6.0.0, there are native threads and no 2GB file size limit.

2.12.5.9 SCO OpenServer 6.0.x Notes

OpenServer 6 includes these key improvements:

- Larger file support up to 1 TB
- Multiprocessor support increased from 4 to 32 processors

- Increased memory support up to 64GB
- Extending the power of UnixWare into OpenServer 6
- Dramatic performance improvement

OpenServer 6.0.0 commands are organized as follows:

- `/bin` is for commands that behave exactly the same as on OpenServer 5.0.x.
- `/u95/bin` is for commands that have better standards conformance, for example Large File System (LFS) support.
- `/udk/bin` is for commands that behave the same as on UnixWare 7.1.4. The default is for the LFS support.

The following is a guide to setting `PATH` on OpenServer 6. If the user wants the traditional OpenServer 5.0.x then `PATH` should be `/bin` first. If the user wants LFS support, the path should be `/u95/bin:/bin`. If the user wants UnixWare 7 support first, the path would be `/udk/bin:/u95/bin:/bin:`.

Use the latest production release of MySQL. Should you choose to use an older release of MySQL on OpenServer 6.0.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

MySQL distribution files with names of the following form are `tar` archives of media are tar archives of media images suitable for installation with the SCO Software Manager (`/etc/custom`) on SCO OpenServer 6:

```
mysql-PRODUCT-4.1.25-sco-osr6-i686.VOLS.tar
```

A distribution where *`PRODUCT`* is `pro-cert` is the Commercially licensed MySQL Pro Certified server. A distribution where *`PRODUCT`* is `pro-gpl-cert` is the MySQL Pro Certified server licensed under the terms of the General Public License (GPL).

Select whichever distribution you wish to install and, after download, extract the `tar` archive into an empty directory. For example:

```
shell> mkdir /tmp/mysql-pro
shell> cd /tmp/mysql-pro
shell> tar xf /tmp/mysql-pro-cert-4.1.25-sco-osr6-i686.VOLS.tar
```

Prior to installation, back up your data in accordance with the procedures outlined in Section 2.11.1, “Upgrading MySQL”.

Remove any previously installed `pkgadd` version of MySQL:

```
shell> pkginfo mysql 2>&1 > /dev/null && pkgrm mysql
```

Install MySQL Pro from media images using the SCO Software Manager:

```
shell> /etc/custom -p SCO:MySQL -i -z /tmp/mysql-pro
```

Alternatively, the SCO Software Manager can be displayed graphically by clicking the `Software Manager` icon on the desktop, selecting `Software -> Install New`, selecting the host, selecting `Media Images` for the Media Device, and entering `/tmp/mysql-pro` as the Image Directory.

After installation, run `mkdev mysql` as the `root` user to configure your newly installed MySQL Pro Certified server.

Note

The installation procedure for VOLS packages does not create the `mysql` user and group that the package uses by default. You should either create the `mysql` user and group, or else select a different user and group using an option in `mkdev mysql`.

If you wish to configure your MySQL Pro server to interface with the Apache Web server using PHP, download and install the PHP update from SCO at ftp://ftp.sco.com/pub/updates/OpenServer/SCOSA-2006.17/.

We have been able to compile MySQL with the following `configure` command on OpenServer 6.0.x:

```
CC=cc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
CXX=CC CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-berkeley-db \
    --with-extra-charsets=complex \
    --build=i686-unknown-sysv5SCO_SV6.0.0
```

If you use `gcc`, you must use `gcc` 2.95.3 or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

The version of Berkeley DB that comes with either UnixWare 7.1.4 or OpenServer 6.0.0 is not used when building MySQL. MySQL instead uses its own version of Berkeley DB. The `configure` command needs to build both a static and a dynamic library in *`src_directory`*`/bdb/build_unix/`, but it does not with MySQL's own `BDB` version. The workaround is as follows.

1. Configure as normal for MySQL.
2. `cd bdb/build_unix/`
3. `cp -p Makefile Makefile.sav`
4. Use same options and run `../dist/configure`.
5. Run `gmake`.
6. `cp -p Makefile.sav Makefile`
7. Change location to the top source directory and run `gmake`.

This enables both the shared and dynamic libraries to be made and work.

SCO provides OpenServer 6 operating system patches at ftp://ftp.sco.com/pub/openserver6.

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenServer.

By default, the maximum file size on a OpenServer 6.0.0 system is 1TB. Some operating system utilities have a limitation of 2GB. The maximum possible file size on UnixWare 7 is 1TB with VXFS or HTFS.

OpenServer 6 can be configured for large file support (file sizes greater than 2GB) by tuning the UNIX kernel.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
```

To make changes to the kernel, use the `idtune name parameter` command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. To set the kernel values, execute the following commands as `root`:

```
# /etc/conf/bin/idtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SFNOLIM 2048
# /etc/conf/bin/idtune HFNOLIM 2048
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

To tune the system, the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `SFNOLIM` and `HFNOLIM` should be at maximum 2048.
- `NPROC` should be set to at least 3000/4000 (depends on number of users).
- The following formulas are recommended to calculate values for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

  ```
  SEMMSL = 13
  ```

  13 is what has been found to be the best for both Progress and MySQL.

  ```
  SEMMNS = SEMMSL * number of db servers to be run on the system
  ```

  Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

  ```
  SEMMNU = SEMMNS
  ```

  Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

2.12.5.10 SCO UnixWare 7.1.x and OpenUNIX 8.0.0 Notes

Use the latest production release of MySQL. Should you choose to use an older release of MySQL on UnixWare 7.1.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

We have been able to compile MySQL with the following `configure` command on UnixWare 7.1.x:

```
CC="cc" CFLAGS="-I/usr/local/include" \
CXX="CC" CXXFLAGS="-I/usr/local/include" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-berkeley-db=./bdb \
    --with-innodb --with-openssl --with-extra-charsets=complex
```

If you want to use `gcc`, you must use `gcc` 2.95.3 or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

The version of Berkeley DB that comes with either UnixWare 7.1.4 or OpenServer 6.0.0 is not used when building MySQL. MySQL instead uses its own version of Berkeley DB. The `configure` command needs to build both a static and a dynamic library in *`src_directory`*`/bdb/build_unix/`, but it does not with MySQL's own `BDB` version. The workaround is as follows.

1. Configure as normal for MySQL.
2. `cd bdb/build_unix/`
3. `cp -p Makefile Makefile.sav`
4. Use same options and run `../dist/configure`.
5. Run `gmake`.
6. `cp -p Makefile.sav Makefile`
7. Change to top source directory and run `gmake`.

This enables both the shared and dynamic libraries to be made and work.

SCO provides operating system patches at ftp://ftp.sco.com/pub/unixware7 for UnixWare 7.1.1, ftp://ftp.sco.com/pub/unixware7/713/ for UnixWare 7.1.3, ftp://ftp.sco.com/pub/unixware7/714/ for UnixWare 7.1.4, and ftp://ftp.sco.com/pub/openunix8 for OpenUNIX 8.0.0.

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenUNIX for OpenUNIX and ftp://ftp.sco.com/pub/security/UnixWare for UnixWare.

The UnixWare 7 file size limit is 1 TB with VXFS. Some OS utilities have a limitation of 2GB.

On UnixWare 7.1.4 you do not need to do anything to get large file support, but to enable large file support on prior versions of UnixWare 7.1.x, run `fsadm`.

```
# fsadm -Fvxfs -o largefiles /
# fsadm /         * Note
# ulimit unlimited
# /etc/conf/bin/idtune SFSZLIM 0x7FFFFFFF     ** Note
# /etc/conf/bin/idtune HFSZLIM 0x7FFFFFFF     ** Note
# /etc/conf/bin/idbuild -B

* This should report "largefiles".
** 0x7FFFFFFF represents infinity for these values.
```

Reboot the system using `shutdown`.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

```
Value           Default         Min             Max
```

```
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
```

To make changes to the kernel, use the `idtune name parameter` command. `idtune` modifies the `/etc/conf/cf.d/stune` file for you. To set the kernel values, execute the following commands as `root`:

```
# /etc/conf/bin/idtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SFNOLIM 2048
# /etc/conf/bin/idtune HFNOLIM 2048
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

To tune the system, the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with `idtune`:

- `SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- `SFNOLIM` and `HFNOLIM` should be at maximum 2048.
- `NPROC` should be set to at least 3000/4000 (depends on number of users).
- The following formulas are recommended to calculate values for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

  ```
  SEMMSL = 13
  ```

  13 is what has been found to be the best for both Progress and MySQL.

  ```
  SEMMNS = SEMMSL * number of db servers to be run on the system
  ```

  Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of database servers (maximum) that you are running on the system at one time.

  ```
  SEMMNU = SEMMNS
  ```

  Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

2.12.6 OS/2 Notes

Note

We no longer test builds on OS/2. The notes in this section are provided for your information but may not work on your system.

MySQL uses quite a few open files. Because of this, you should add something like the following to your `CONFIG.SYS` file:

```
```

```
SET EMXOPT=-c -n -h1024
```

If you do not do this, you may encounter the following error:

```
File 'xxxx' not found (Errcode: 24)
```

When using MySQL with OS/2 Warp 3, FixPack 29 or above is required. With OS/2 Warp 4, FixPack 4 or above is required. This is a requirement of the Pthreads library. MySQL must be installed on a partition with a type that supports long file names, such as HPFS, FAT32, and so on.

The `INSTALL.CMD` script must be run from OS/2's own `CMD.EXE` and may not work with replacement shells such as `4OS2.EXE`.

The `scripts/mysql-install-db` script has been renamed. It is called `install.cmd` and is a REXX script, which sets up the default MySQL security settings and creates the WorkPlace Shell icons for MySQL.

Dynamic module support is compiled in but not fully tested. Dynamic modules should be compiled using the Pthreads runtime library.

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrtl -I../include -I../regex -I.. \
    -o example udf_example.cc -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Note

Due to limitations in OS/2, UDF module name stems must not exceed eight characters. Modules are stored in the `/mysql2/udf` directory; the `safe-mysqld.cmd` script puts this directory in the `BEGINLIBPATH` environment variable. When using UDF modules, specified extensions are ignored---it is assumed to be `.udf`. For example, in Unix, the shared module might be named `example.so` and you would load a function from it like this:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example.so';
```

In OS/2, the module would be named `example.udf`, but you would not specify the module extension:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

2.13 Environment Variables

This section lists all the environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables.

In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See Section 4.2.3.3, “Using Option Files”.

Variable	Description
`CXX`	The name of your C++ compiler (for running `configure`).
`CC`	The name of your C compiler (for running `configure`).

Variable	Description
CFLAGS	Flags for your C compiler (for running configure).
CXXFLAGS	Flags for your C++ compiler (for running configure).
DBI_USER	The default user name for Perl DBI.
DBI_TRACE	Trace options for Perl DBI.
HOME	The default path for the mysql history file is $HOME/.mysql_history.
LD_RUN_PATH	Used to specify the location of libmysqlclient.so.
MYSQL_DEBUG	Debug trace options when debugging.
MYSQL_GROUP_SUFFIX	Option group suffix value (like specifying --defaults-group-suffix).
MYSQL_HISTFILE	The path to the mysql history file. If this variable is set, its value overrides the default for $HOME/.mysql_history.
MYSQL_HOME	The path to the directory in which the server-specific my.cnf file resides (as of MySQL 5.0.3).
MYSQL_HOST	The default host name used by the mysql command-line client.
MYSQL_PS1	The command prompt to use in the mysql command-line client.
MYSQL_PWD	The default password when connecting to mysqld. Note that using this is insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”.
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_UNIX_PORT	The default Unix socket file name; used for connections to localhost.
PATH	Used by the shell to find MySQL programs.
TMPDIR	The directory where temporary files are created.
TZ	This should be set to your local time zone. See Section B.5.4.6, “Time Zone Problems”.
UMASK	The user-file creation mode when creating files. See note following table.
UMASK_DIR	The user-directory creation mode when creating directories. See note following table.
USER	The default user name on Windows and NetWare when connecting to mysqld.

For information about the mysql history file, see Section 4.5.1.3, “mysql Logging”.

The UMASK and UMASK_DIR variables, despite their names, are used as modes, not masks:

- If UMASK is set, mysqld uses ($UMASK | 0600) as the mode for file creation, so that newly created files have a mode in the range from 0600 to 0666 (all values octal).
- If UMASK_DIR is set, mysqld uses ($UMASK_DIR | 0700) as the base mode for directory creation, which then is AND-ed with ~(~$UMASK & 0666), so that newly created directories have a mode in the range from 0700 to 0777 (all values octal). The AND operation may remove read and write permissions from the directory mode, but not execute permissions.

In MySQL 3.23.25 and above, MySQL assumes that the value for UMASK or UMASK_DIR is in octal if it starts with a zero.

2.14 Perl Installation Notes

Perl support for MySQL is provided by means of the DBI/DBD client interface. The interface requires Perl 5.6.0, and 5.6.1 or later is preferred. DBI *does not work* if you have an older version of Perl.

To use transactions with Perl DBI, you must use `DBD::mysql` 2.0900 or newer. To use the MySQL 4.1 or newer client library, you must use `DBD::mysql` 2.9003 or newer. Support for server-side prepared statements requires `DBD::mysql` 3.0009 or newer. Current versions of `DBD::mysql` on CPAN are 4.xxxx or higher and support all these capabilities.

As of MySQL 3.22.8, Perl support is no longer included with MySQL distributions. You can obtain the necessary modules from http://search.cpan.org for Unix, or by using the ActiveState `ppm` program on Windows. The following sections describe how to do this.

Perl support for MySQL must be installed if you want to run the MySQL benchmark scripts; see Section 7.1.3, “The MySQL Benchmark Suite”. It is also required for the MySQL Cluster `ndb_size.pl` utility; see Section 15.4.18, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”.

2.14.1 Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. However, if you installed MySQL from RPM files on Linux, be sure that you've installed the developer RPM. The client programs are in the client RPM, but client programming support is in the developer RPM.

If you want to install Perl support, the files you need can be obtained from the CPAN (Comprehensive Perl Archive Network) at http://search.cpan.org.

The easiest way to install Perl modules on Unix is to use the `CPAN` module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The `DBD::mysql` installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and `ODBC` on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

`DBI` requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed `tar` archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

   ```
   shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd DBI-VERSION
   ```

3. Build the distribution and compile everything:

   ```
   shell> perl Makefile.PL
   shell> make
   ```

```
shell> make test
shell> make install
```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL, particularly if you notice symptoms such as that all your `DBI` scripts fail after you upgrade MySQL.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: http://servers.digitaldaze.com/extensions/perl/modules.html#modules

Look under the heading “Installing New Modules that Require Locally Installed Modules.”

2.14.2 Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

1. Get ActiveState Perl from http://www.activestate.com/Products/ActivePerl/ and install it.
2. Open a console window (a “DOS window”).
3. If necessary, set the `HTTP_proxy` variable. For example, you might try a setting like this:

   ```
   set HTTP_proxy=my.proxy.com:3128
   ```

4. Start the PPM program:

   ```
   shell> C:\perl\bin\ppm.pl
   ```

5. If you have not previously done so, install `DBI`:

   ```
   ppm> install DBI
   ```

6. If this succeeds, run the following command:

   ```
   ppm> install DBD-mysql
   ```

This procedure should work with ActiveState Perl 5.6 or newer.

If you cannot get the procedure to work, you should install the MyODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.14.3 Problems Using the Perl `DBI`/`DBD` Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Compile the DBD::mysql distribution with perl Makefile.PL -static -config rather than perl Makefile.PL.
- Copy libmysqlclient.so to the directory where your other shared libraries are located (probably /usr/lib or /lib).
- Modify the -L options used to compile DBD::mysql to reflect the actual location of libmysqlclient.so.
- On Linux, you can add the path name of the directory where libmysqlclient.so is located to the /etc/ld.so.conf file.
- Add the path name of the directory where libmysqlclient.so is located to the LD_RUN_PATH environment variable. Some systems use LD_LIBRARY_PATH instead.

Note that you may also need to modify the -L options if there are other libraries that the linker fails to find. For example, if the linker cannot find libc because it is in /lib and the link command specifies -L/usr/lib, change the -L option to -L/lib or add -L/lib to the existing link command.

If you get the following errors from DBD::mysql, you are probably using gcc (or using an old binary compiled with gcc):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add -L/usr/lib/gcc-lib/... -lgcc to the link command when the mysql.so library gets built (check the output from make for mysql.so when you compile the Perl client). The -L option should specify the path name of the directory where libgcc.a is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with gcc. In this case, you can solve the mismatch by compiling both with gcc.

You may see the following error from DBD::mysql when you run the tests:

```
t/00base............install_driver(mysql) failed:
Can't load '../blib/arch/auto/DBD/mysql/mysql.so' for module DBD::mysql:
../blib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 169.
```

This means that you need to include the -lz compression library on the link line. That can be done by changing the following line in the file lib/DBD/mysql/Install.pm:

```
$sysliblist .= " -lm";
```

Change that line to:

```
$sysliblist .= " -lm -lz";
```

After this, you *must* run make realclean and then proceed with the installation from the beginning.

If you want to install DBI on SCO, you have to edit the Makefile in DBI-*xxx* and each subdirectory. Note that the following assumes gcc 2.95.2 or newer:

```
OLD:                                NEW:
CC = cc                             CC = gcc
```

```
CCCDLFLAGS = -KPIC -W1,-Bexport       CCCDLFLAGS = -fpic
CCDLFLAGS = -wl,-Bexport              CCDLFLAGS =

LD = ld                               LD = gcc -G -fpic
LDDLFLAGS = -G -L/usr/local/lib       LDDLFLAGS = -L/usr/local/lib
LDFLAGS = -belf -L/usr/local/lib      LDFLAGS = -L/usr/local/lib

LD = ld                               LD = gcc -G -fpic
OPTIMISE = -Od                        OPTIMISE = -O1

OLD:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

NEW:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include
```

These changes are necessary because the Perl dynaloader does not load the DBI modules if they were compiled with icc or cc.

If you want to use the Perl module on a system that does not support dynamic linking (such as SCO), you can generate a static version of Perl that includes DBI and DBD::mysql. The way this works is that you generate a version of Perl with the DBI code linked in and install it on top of your current Perl. Then you use that to build a version of Perl that additionally has the DBD code linked in, and install that.

On SCO, you must have the following environment variables set:

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

Or:

```
LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
    /usr/skunk/man:
```

First, create a Perl that includes a statically linked DBI module by running these commands in the directory where your DBI distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Then, you must install the new Perl. The output of make perl indicates the exact make command you need to execute to perform the installation. On SCO, this is make -f Makefile.aperl inst_perl MAP_TARGET=perl.

Next, use the just-created Perl to create another Perl that also includes a statically linked DBD::mysql by running these commands in the directory where your DBD::mysql distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Finally, you should install this new Perl. Again, the output of make perl indicates the command to use.

Chapter 3 Tutorial

Table of Contents

This chapter provides a tutorial introduction to MySQL by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that enables you to connect to a MySQL server, run queries, and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help` [260] option:

```
shell> mysql --help
```

This chapter assumes that `mysql` is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as Chapter 5, *MySQL Server Administration*.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1 Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you will

also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

`host` and `user` represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The `********` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 4.1.25-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The `mysql>` prompt tells you that `mysql` is ready for you to enter commands.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as `ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)`, it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of Chapter 2, *Installing and Upgrading MySQL* that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see Section B.5.2, “Common Errors When Using MySQL Programs”.

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` (or `\q`) at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control-D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2 Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that is okay. At this point, it is more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how `mysql` works.

Here is a simple command that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+------------+--------------+
| VERSION()  | CURRENT_DATE |
+------------+--------------+
| 4.1.14-Max | 2005-09-03   |
+------------+--------------+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A command normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a command, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another command.
- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.
- `mysql` shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the “rows in set” line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here is another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
```

```
+------------+
| VERSION()  |
+------------+
| 4.1.14-Max |
+------------+

+---------------------+
| NOW()               |
+---------------------+
| 2005-09-03 12:27:16 |
+---------------------+
```

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here is a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2005-09-03   |
+---------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a command that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in.

Prompt	Meaning
`mysql>`	Ready for new command.
`->`	Waiting for next line of multiple-line command.
`'>`	Waiting for next line, waiting for completion of a string that began with a single quote (“`'`”).
`">`	Waiting for next line, waiting for completion of a string that began with a double quote (“`"`”).
`` `> ``	Waiting for next line, waiting for completion of an identifier that began with a backtick (“`` ` ``”).
`/*>`	Waiting for next line, waiting for completion of a comment that began with `/*`.

The `` `> `` prompt was implemented MySQL 4.0.16. The `/*>` prompt was implemented in MySQL 4.1.12.

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either "`'`" or "`"`" characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a "`'`" or "`"`" quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quotation mark.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new command.

The `` `> `` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and `` `> `` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql`—including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current command.

3.3 Creating and Using a Database

Once you know how to enter commands, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL Web site. It is available in both compressed `tar` file and Zip formats at http://dev.mysql.com/doc/.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The `mysql` database describes user access privileges. The `test` database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; `SHOW DATABASES` does not show databases that you have no privileges for if you do not have the `SHOW DATABASES` privilege. See Section 12.4.5.8, “`SHOW DATABASES` Syntax”.

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

`USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where `your_mysql_name` is the MySQL user name assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1 Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE`, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as `ERROR 1044 (42000): Access denied for user 'monty'@'localhost' to database 'menagerie'` when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see Section 5.5, “The MySQL Access Privilege System”.

Creating a database does not select it for use; you must do that explicitly. To make `menagerie` the current database, use this command:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a `mysql` session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke `mysql`. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Important

`menagerie` in the command just shown is **not** your password. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-pmypassword`, not as `-p mypassword`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

Note

You can see at any time which database is currently selected using `SELECT DATABASE()` [872].

3.3.2 Creating a Table

Creating the database is the easy part, but at this point it is empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
```

```
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can pick any length from `1` to `255`, whatever seems most reasonable to you. (If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.)

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+----------------------+
| Tables in menagerie |
+----------------------+
| pet                  |
+----------------------+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
```

```
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see Chapter 10, *Data Types*.

3.3.3 Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may be different from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler        Gwen    bird    \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

If you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use this statement instead:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

(On an Apple machine running OS X, you would likely want to use `LINES TERMINATED BY '\r'`.)

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See Section 5.4.5, “Security Issues with `LOAD DATA LOCAL`”, for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

String and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4 Retrieving Information from a Table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

`what_to_select` indicates what you want to see. This can be a list of columns, or `*` to indicate “all columns.” *`which_table`* indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it is present, *`conditions_to_satisfy`* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1 Selecting All Data

The simplest form of `SELECT` retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of SELECT is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file pet.txt to correct the error, then empty the table and reload it using DELETE and LOAD DATA:

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an UPDATE statement:

```
mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
```

  The UPDATE changes only the record in question and does not require you to reload the table.

3.3.4.2 Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the WHERE clause from the SELECT statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as 'bowser', 'BOWSER', and so forth. The query result is the same.

You can specify conditions on any column, not just name. For example, if you want to know which animals were born during or after 1998, test the birth column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the `AND` [787] logical operator. There is also an `OR` [787] operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

`AND` [787] and `OR` [787] may be intermixed, although `AND` [787] has higher precedence than `OR` [787]. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
    -> OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3 Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
```

```
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4 Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` [859] like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
    -> ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

The `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5 Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, compute the difference in the year part of the current date and the birth date, then subtract one if the current date occurs earlier in the calendar year than the birth date. The following query shows, for each pet, the birth date, the current date, and the age in years.

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
```

```
    -> FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

Here, `YEAR()` [844] pulls out the year part of a date and `RIGHT()` [800] pulls off the rightmost five characters that represent the `MM-DD` (calendar year) part of the date. The part of the expression that compares the `MM-DD` values evaluates to 1 or 0, which adjusts the year difference down a year if `CURDATE()` [828] occurs earlier in the year than `birth`. The full expression is somewhat ungainly, so an *alias* (`age`) is used to make the output column label more meaningful.

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
    -> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See Section 3.3.4.6, “Working with `NULL` Values”.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()` [844], `MONTH()` [837], and `DAYOFMONTH()` [833]. `MONTH()` [837] is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)` [837]:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is `4` and you can look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

There is a small complication if the current month is December. You cannot merely add one to the month number (`12`) and look for animals born in month `13`, because there is no such month. Instead, you look for animals born in January (month `1`).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` [829] enables you to add a time interval to a given date. If you add a month to the value of `CURDATE()` [828], then extract the month part with `MONTH()` [837], the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
```

```
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add `1` to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to `0` if it is currently `12`:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()` [837] returns a number between `1` and `12`. And `MOD(something,12)` [821] returns a number between `0` and `11`. So the addition has to be after the `MOD()` [821], otherwise we would go from November (`11`) to January (`1`).

3.3.4.6 Working with `NULL` Values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means “a missing unknown value” and it is treated somewhat differently from other values. To test for `NULL`, you cannot use the arithmetic comparison operators such as `=`, `<`, or `<>`. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Clearly you get no meaningful results from these comparisons. Use the `IS NULL` [783] and `IS NOT NULL` [783] operators instead:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

In MySQL, `0` or `NULL` means false and anything else means true. The default truth value from a boolean operation is `1`.

This special treatment of `NULL` is why, in the previous section, it was necessary to determine which animals are no longer alive using `death IS NOT NULL` instead of `death <> NULL`.

Two `NULL` values are regarded as equal in a `GROUP BY`.

When doing an `ORDER BY`, `NULL` values are presented first if you do `ORDER BY ... ASC` and last if you do `ORDER BY ... DESC`.

Note that MySQL 4.0.2 to 4.0.10 incorrectly always sorts `NULL` values first regardless of the sort direction.

A common error when working with `NULL` is to assume that it is not possible to insert a zero or an empty string into a column defined as `NOT NULL`, but this is not the case. These are in fact values, whereas `NULL` means “not having a value.” You can test this easily enough by using `IS [NOT] NULL` as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+---------------+------------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+---------------+------------+----------------+
|         0 |             1 |          0 |              1 |
+-----------+---------------+------------+----------------+
```

```
+----------+--------------+-----------+---------------+
```

Thus it is entirely possible to insert a zero or empty string into a `NOT NULL` column, as these are in fact `NOT NULL`. See Section B.5.5.3, “Problems with `NULL` Values”.

3.3.4.7 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as `vi`, `grep`, and `sed`.

SQL pattern matching enables you to use “`_`” to match any single character and “`%`” to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. You do not use `=` or `<>` when you use SQL patterns; use the `LIKE` [804] or `NOT LIKE` [806] comparison operators instead.

To find names beginning with “`b`”:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with “`fy`”:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a “`w`”:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use five instances of the “`_`” pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the `REGEXP` [808] and `NOT REGEXP` [808] operators (or `RLIKE` [808] and `NOT RLIKE` [808], which are synonyms).

The following list describes some characteristics of extended regular expressions:

- "`.`" matches any single character.
- A character class "`[...]`" matches any character within the brackets. For example, "`[abc]`" matches "`a`", "`b`", or "`c`". To name a range of characters, use a dash. "`[a-z]`" matches any letter, whereas "`[0-9]`" matches any digit.
- "`*`" matches zero or more instances of the thing preceding it. For example, "`x*`" matches any number of "`x`" characters, "`[0-9]*`" matches any number of digits, and "`.*`" matches any number of anything.
- A `REGEXP` [808] pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a `LIKE` [804] pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use "`^`" at the beginning or "`$`" at the end of the pattern.

To demonstrate how extended regular expressions work, the `LIKE` [804] queries shown previously are rewritten here to use `REGEXP` [808].

To find names beginning with "`b`", use "`^`" to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

Prior to MySQL 3.23.4, `REGEXP` [808] is case sensitive, and the previous query will return no rows. In this case, to match either lowercase or uppercase "`b`", use this query instead:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^[bB]';
```

From MySQL 3.23.4 on, if you really want to force a `REGEXP` [808] comparison to be case sensitive, use the `BINARY` [859] keyword to make one of the strings a binary string. This query matches only lowercase "`b`" at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with "`fy`", use "`$`" to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a "`w`", use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
```

```
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use "^" and "$" to match the beginning and end of the name, and five instances of "." in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the {*n*} ("repeat-*n*-times") operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

Section 11.5.2, "Regular Expressions", provides more information about the syntax for regular expressions.

3.3.4.8 Counting Rows

Databases are often used to answer the question, "How often does a certain type of data occur in a table?" For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as "How many rows are in the `pet` table?" because there is one record per pet. `COUNT(*)` [882] counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` [882] if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
```

```
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

The preceding query uses `GROUP BY` to group all records for each `owner`. The use of `COUNT()` [882] in conjunction with `GROUP BY` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()` [882]. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = 'dog' OR species = 'cat'
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
```

```
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE sex IS NOT NULL
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

If you name columns to select in addition to the `COUNT()` [882] value, a `GROUP BY` clause should be present that names those same columns. Otherwise, an error occurs:

```
mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140: Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

3.3.4.9 Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs to contain the following information:

- The pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it is easiest to load the initial records by creating a tab-delimited text file containing the following information.

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib

name	date	type	remark
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
    -> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
    -> remark
    -> FROM pet INNER JOIN event
    ->   ON pet.name = event.name
    -> WHERE event.type = 'litter';
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

There are several things to note about this query:

- The `FROM` clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses `ON` clause to match up records in the two tables based on the `name` values.

  The query uses an `INNER JOIN` to combine the tables. An `INNER JOIN` permits rows from either table to appear in the result if and only if both tables meet the conditions specified in the `ON` clause. In this example, the `ON` clause specifies that the `name` column in the `pet` table must match the `name` column in the `event` table. If a name appears in one table but not the other, the row will not appear in the result because the condition in the `ON` clause fails.
- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to produce candidate pairs of males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1 INNER JOIN pet AS p2
    ->   ON p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm';
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
+--------+------+--------+------+---------+
```

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4 Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` [872] function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you have not yet selected any database, the result is `NULL` (or the empty string before MySQL 4.1.1).

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_menagerie |
+---------------------+
| event               |
| pet                 |
+---------------------+
```

The name of the column in the output produced by this statement is always `Tables_in_`*`db_name`*, where *`db_name`* is the name of the database. See Section 12.4.5.24, “`SHOW TABLES` Syntax”, for more information.

If you want to find out about the structure of a table, the `DESCRIBE` statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
```

```
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value. `Extra` displays special information about columns: If a column was created with the `AUTO_INCREMENT` option, the value will be `auto_increment` rather than empty.

`DESC` is a short form of `DESCRIBE`. See Section 12.7.1, “`DESCRIBE` Syntax”, for more information.

You can obtain the `CREATE TABLE` statement necessary to create an existing table using the `SHOW CREATE TABLE` statement. See Section 12.4.5.7, “`SHOW CREATE TABLE` Syntax”.

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them. See Section 12.4.5.13, “`SHOW INDEX` Syntax”, for more about this statement.

3.5 Using `mysql` in Batch Mode

In the previous sections, you used `mysql` interactively to enter queries and view the results. You can also run `mysql` in batch mode. To do this, put the commands you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` [261] command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script enables you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line commands or multiple-statement sequences of commands. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

```
shell> mysql < batch-file | more
```

- You can catch the output in a file for further processing:

```
shell> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so that they can also run the commands.
- Some situations do not allow for interactive use, for example, when you run a query from a `cron` job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the commands that are executed, use `mysql -vvv`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See Section 4.5.1.5, “Executing SQL Statements from a Text File”, for more information.

3.6 Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then (`article`, `dealer`) is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
shell> mysql your-database-name
```

(In most MySQL installations, you can use the database named `test`).

You can create and populate the example table with these statements:

```
CREATE TABLE shop (
    article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
    price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
    (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.6.1 The Maximum Value for a Column

“What is the highest item number?”

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2 The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Other solutions are to use a `LEFT JOIN` or to sort all rows descending by price and get only the first row using the MySQL-specific `LIMIT` clause:

```
SELECT s1.article, s1.dealer, s1.price
```

```
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note

If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them.

3.6.3 Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

In standard SQL (and as of MySQL 4.1), the problem can be solved with a subquery like this:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article);
```

The preceding example uses a correlated subquery, which can be inefficient (see Section 12.2.8.7, “Correlated Subqueries”). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause or a `LEFT JOIN`.

Uncorrelated subquery:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price;

SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
```

```
WHERE s2.article IS NULL;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and the `s2` rows values will be `NULL`. See Section 12.2.7.1, “`JOIN` Syntax”.

Before MySQL 4.1, subqueries are unavailable. Another approach is to solve the problem in several steps:

1. Get the list of (article,maxprice) pairs.
2. For each article, get the corresponding rows that have the stored maximum price.

This can easily be done with a temporary table and a join:

`LEFT JOIN`:

```
CREATE TEMPORARY TABLE tmp (
        article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
        price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL);

LOCK TABLES shop READ;

INSERT INTO tmp SELECT article, MAX(price) FROM shop GROUP BY article;

SELECT shop.article, dealer, shop.price FROM shop, tmp
WHERE shop.article=tmp.article AND shop.price=tmp.price;

UNLOCK TABLES;

DROP TABLE tmp;
```

If you don't use a `TEMPORARY` table, you must also lock the `tmp` table.

“Can it be done with a single query?”

Yes, but only by using a quite inefficient trick called the “MAX-CONCAT trick”:

```
SELECT article,
       SUBSTRING( MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 7) AS dealer,
  0.00+LEFT(      MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 6) AS price
FROM   shop
GROUP BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The last example can be made a bit more efficient by doing the splitting of the concatenated column in the client.

3.6.5 Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See Section 8.4, “User-Defined Variables”.)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.6.6 Using Foreign Keys

In MySQL 3.23.44 and up, `InnoDB` tables support checking of foreign key constraints. See Section 13.2, “The `InnoDB` Storage Engine”, and Section 1.9.5.6, “Foreign Keys”.

A foreign key constraint is not required merely to join two tables. For storage engines other than `InnoDB`, it is possible when defining a column to use a `REFERENCES` *`tbl_name(col_name)`* clause, which has no actual effect, and *serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table*. It is extremely important to realize when using this syntax that:

- MySQL does not perform any sort of `CHECK` to make sure that *`col_name`* actually exists in *`tbl_name`* (or even that *`tbl_name`* itself exists).
- MySQL does not perform any sort of action on *`tbl_name`* such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no `ON DELETE` or `ON UPDATE` behavior whatsoever. (Although you can write an `ON DELETE` or `ON UPDATE` clause as part of the `REFERENCES` clause, it is also ignored.)
- This syntax creates a *column*; it does **not** create any sort of index or key.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
```

```
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM person p INNER JOIN shirt s
   ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

When used in this fashion, the `REFERENCES` clause is not displayed in the output of `SHOW CREATE TABLE` or `DESCRIBE`:

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

The use of `REFERENCES` in this way as a comment or “reminder” in a column definition works with both `MyISAM` and `BerkeleyDB` tables.

3.6.7 Searching on Two Keys

An `OR` [787] using a single key is well optimized, as is the handling of `AND` [787].

The one tricky case is that of searching on two different keys combined with `OR` [787]:

```
SELECT field1_index, field2_index FROM test_table
```

```
WHERE field1_index = '1' OR  field2_index = '1'
```

In MySQL 4.0 and up, you can solve the problem efficiently by using a `UNION` that combines the output of two separate `SELECT` statements. See Section 12.2.7.3, “`UNION` Syntax”.

Each `SELECT` searches only one key and can be optimized:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

Prior to MySQL 4.0, you can achieve the same effect by using a `TEMPORARY` table and separate `SELECT` statements. This type of optimization is also very good if you are using very complicated queries where the SQL server does the optimizations in the wrong order.

```
CREATE TEMPORARY TABLE tmp
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1';
INSERT INTO tmp
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
SELECT * from tmp;
DROP TABLE tmp;
```

This method of solving the problem is in effect a `UNION` of two queries.

3.6.8 Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
             day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
            (2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Which returns:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9 Using `AUTO_INCREMENT`

The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
) ENGINE=MyISAM;

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

No value was specified for the AUTO_INCREMENT column, so MySQL assigned sequence numbers automatically. You can also explicitly assign NULL or 0 to the column to generate sequence numbers.

You can retrieve the most recent AUTO_INCREMENT value with the LAST_INSERT_ID() [874] SQL function or the mysql_insert_id() C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Use a large enough integer data type for the AUTO_INCREMENT column to hold the maximum sequence value you will need. When the column reaches the upper limit of the data type, the next attempt to generate a sequence number fails. For example, if you use TINYINT, the maximum permissible sequence number is 127. For TINYINT UNSIGNED, the maximum is 255.

Note

For a multiple-row insert, LAST_INSERT_ID() [874] and mysql_insert_id() actually return the AUTO_INCREMENT key from the *first* of the inserted rows. This enables multiple-row inserts to be reproduced correctly on other servers in a replication setup.

For MyISAM and BDB tables you can specify AUTO_INCREMENT on a secondary column in a multiple-column index. In this case, the generated value for the AUTO_INCREMENT column is calculated as MAX(*auto_increment_column*) + 1 WHERE prefix=*given-prefix* [884]. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (
    grp ENUM('fish','mammal','bird') NOT NULL,
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
```

```
    ('mammal','dog'),('mammal','cat'),
    ('bird','penguin'),('fish','lax'),('mammal','whale'),
    ('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

In this case (when the `AUTO_INCREMENT` column is part of a multiple-column index), `AUTO_INCREMENT` values are reused if you delete the row with the biggest `AUTO_INCREMENT` value in any group. This happens even for `MyISAM` tables, for which `AUTO_INCREMENT` values normally are not reused.

If the `AUTO_INCREMENT` column is part of multiple indexes, MySQL will generate sequence values using the index that begins with the `AUTO_INCREMENT` column, if there is one. For example, if the `animals` table contained indexes `PRIMARY KEY (grp, id)` and `INDEX (id)`, MySQL would ignore the `PRIMARY KEY` for generating sequence values. As a result, the table would contain a single sequence, not a sequence per `grp` value.

To start with an `AUTO_INCREMENT` value other than 1, you can set that value with `CREATE TABLE` or `ALTER TABLE`, like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

Note that this feature is available for `InnoDB` tables only as of MySQL 4.1.12.

More information about `AUTO_INCREMENT` is available here:

- How to assign the `AUTO_INCREMENT` attribute to a column: Section 12.1.5, “`CREATE TABLE` Syntax”, and Section 12.1.2, “`ALTER TABLE` Syntax”.
- How `AUTO_INCREMENT` behaves depending on the `NO_AUTO_VALUE_ON_ZERO` [458] SQL mode: Section 5.1.6, “Server SQL Modes”.
- How to use the `LAST_INSERT_ID()` [874] function to find the row that contains the most recent `AUTO_INCREMENT` value: Section 11.13, “Information Functions”.
- Setting the `AUTO_INCREMENT` value to be used: Section 5.1.3, “Server System Variables”.
- `AUTO_INCREMENT` and replication: Chapter 14, *Replication*.

3.7 Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the `LogFormat` line writes to the log file.

Chapter 4 MySQL Programs

Table of Contents

This chapter provides a brief overview of the MySQL command-line programs provided by Oracle Corporation. It also discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Finally, the chapter provides more detailed descriptions of individual programs, including which options they recognize.

4.1 Overview of MySQL Programs

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one, with the exception of MySQL Cluster programs. Each program's description indicates its invocation syntax and the options that it supports. Chapter 15, *MySQL Cluster*, describes programs specific to MySQL Cluster.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, *Installing and Upgrading MySQL*, for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysql --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See Section 4.2, “Using MySQL Programs”, for general information on invoking programs and specifying program options.

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

- `mysqld`

  The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See Section 4.3.1, “`mysqld` — The MySQL Server”.

- `mysqld-max`

  A version of the server that includes additional features. See Section 5.2, “The `mysqld-max` Extended MySQL Server”.

- `mysqld_safe`

  A server startup script. `mysqld_safe` attempts to start `mysqld-max` if it exists, and `mysqld` otherwise. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

- `mysql.server`

A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

- `mysqld_multi`

  A server startup script that can start or stop multiple servers installed on the system. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

There are several programs that perform setup operations during MySQL installation or upgrading:

- `comp_err`

  This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”.

- `make_binary_distribution`

  This program makes a binary release of a compiled MySQL. This could be sent by FTP to `/pub/mysql/upload/` on `ftp.mysql.com` for the convenience of other MySQL users.

- `mysql_create_system_tables`

  This script is invoked by `mysql_install_db` to generate the SQL statements required to initialize the grant tables with default privileges. See Section 4.4.3, “`mysql_create_system_tables` — Generate Statements to Initialize MySQL System Tables”.

- `mysql_fix_privilege_tables`

  This program is used after a MySQL upgrade operation. It updates the grant tables with any changes that have been made in newer versions of MySQL. See Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

- `mysql_install_db`

  This script creates the MySQL database and initializes the grant tables with default privileges. It is usually executed only once, when first installing MySQL on a system. See Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”, Section 2.10.2, “Unix Postinstallation Procedures”, and Section 4.4.6, “`mysql_install_db` — Initialize MySQL Data Directory”.

- `mysql_secure_installation`

  This program enables you to improve the security of your MySQL installation. SQL. See Section 4.4.7, “`mysql_secure_installation` — Improve MySQL Installation Security”.

- `mysql_tzinfo_to_sql`

  This program loads the time zone tables in the `mysql` database using the contents of the host system *zoneinfo* database (the set of files describing time zones). SQL. See Section 4.4.8, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”.

- `make_win_src_distribution`

  This program is used on Unix or Unix-like systems to create a MySQL source distribution that can be compiled on Windows. See Section 2.9.7.2, “Creating a Windows Source Package from the Latest Development Source”, and Section 4.4.2, “`make_win_src_distribution` — Create Source Distribution for Windows”.

MySQL client programs:

- `mysql`

  The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”.

- `mysqladmin`

  A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. `mysqladmin` can also be used to retrieve version, process, and status information from the server. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

- `mysqlcheck`

  A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”.

- `mysqldump`

  A client that dumps a MySQL database into a file as SQL, text, or XML. See Section 4.5.4, “`mysqldump` — A Database Backup Program”.

- `mysqlimport`

  A client that imports text files into their respective tables using `LOAD DATA INFILE`. See Section 4.5.5, “`mysqlimport` — A Data Import Program”.

- `mysqlshow`

  A client that displays information about databases, tables, columns, and indexes. See Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

MySQL administrative and utility programs:

- `myisam_ftdump`

  A utility that displays information about full-text indexes in `MyISAM` tables. See Section 4.6.1, “`myisam_ftdump` — Display Full-Text Index information”.

- `myisamchk`, `isamchk`

  A utility to describe, check, optimize, and repair `MyISAM` tables. `isamchk` is a similar program for `ISAM` tables. See Section 4.6.2, “`myisamchk` — MyISAM Table-Maintenance Utility”.

- `myisamlog`, `isamlog`

  Utilities that process the contents of a `MyISAM` or `ISAM` log file. See Section 4.6.3, “`myisamlog` — Display MyISAM Log File Contents”.

- `myisampack`, `pack_isam`

  Utilities that compress `MyISAM` or `ISAM` tables to produce smaller read-only tables. See Section 4.6.4, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.

- `mysqlaccess`

A script that checks the access privileges for a host name, user name, and database combination. See Section 4.6.5, “`mysqlaccess` — Client for Checking Access Privileges”.

- `mysqlbinlog`

  A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See Section 4.6.6, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

- `mysqldumpslow`

  A utility to read and summarize the contents of a slow query log. See Section 4.6.7, “`mysqldumpslow` — Summarize Slow Query Log Files”.

- `mysqlhotcopy`

  A utility that quickly makes backups of `MyISAM` or `ISAM` tables while the server is running. See Section 4.6.8, “`mysqlhotcopy` — A Database Backup Program”.

- `mysql_convert_table_format`

  A utility that converts tables in a database to use a given storage engine. See Section 4.6.11, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”.

- `mysql_explain_log`

  A utility that analyzes queries in the MySQL query log using `EXPLAIN` See Section 4.6.12, “`mysql_explain_log` — Use EXPLAIN on Statements in Query Log”.

- `mysql_find_rows`

  A utility that reads files containing SQL statements (such as update logs) and extracts statements that match a given regular expression. See Section 4.6.13, “`mysql_find_rows` — Extract SQL Statements from Files”.

- `mysql_fix_extensions`

  A utility that converts the extensions for `MyISAM` (or `ISAM`) table files to lowercase. This can be useful after transferring the files from a system with case-insensitive file names to a system with case-sensitive file names. See Section 4.6.14, “`mysql_fix_extensions` — Normalize Table File Name Extensions”.

- `mysql_setpermission`

  A utility for interactively setting permissions in the MySQL grant tables. See Section 4.6.15, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”.

- `mysql_tableinfo`

  A utility that generates database metadata. Section 4.6.16, “`mysql_tableinfo` — Generate Database Metadata”.

- `mysql_waitpid`

  A utility that kills the process with a given process ID. See Section 4.6.17, “`mysql_waitpid` — Kill Process and Wait for Its Termination”.

- `mysql_zap`

A utility that kills processes that match a pattern. See Section 4.6.18, “`mysql_zap` — Kill Processes That Match a Pattern”.

MySQL program-development utilities:

- `msql2mysql`

  A shell script that converts `mSQL` programs to MySQL. It doesn't handle every case, but it gives a good start when converting. See Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”.

- `mysql_config`

  A shell script that produces the option values needed when compiling MySQL programs. See Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”.

- `my_print_defaults`

  A utility that shows which options are present in option groups of option files. See Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”.

- `resolve_stack_dump`

  A utility program that resolves a numeric stack trace dump to symbols. See Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”.

Miscellaneous utilities:

- `perror`

  A utility that displays the meaning of system or MySQL error codes. See Section 4.8.1, “`perror` — Explain Error Codes”.

- `replace`

  A utility program that performs string replacement in the input text. See Section 4.8.2, “`replace` — A String-Replacement Utility”.

- `resolveip`

  A utility program that resolves a host name to an IP address or vice versa. See Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”.

Oracle Corporation also provides several GUI tools for administering and otherwise working with MySQL Server:

- MySQL Workbench: This is the latest graphical tool for working with MySQL databases.
- MySQL Administrator: This tool is used for administering MySQL servers, databases, tables, and user accounts.
- MySQL Query Browser: This graphical tool is used for creating, executing, and optimizing queries on MySQL databases.
- MySQL Migration Toolkit: This tool helps you migrate schemas and data from other relational database management systems for use with MySQL.

These GUI programs are available at http://dev.mysql.com/downloads/. Each has its own manual that you can access at http://dev.mysql.com/doc/.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables.

Environment Variable	Meaning
`MYSQL_UNIX_PORT`	The default Unix socket file; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default port number; used for TCP/IP connections
`MYSQL_PWD`	The default password
`MYSQL_DEBUG`	Debug trace options when debugging
`TMPDIR`	The directory where temporary tables and files are created

For a full list of environment variables used by MySQL programs, see Section 2.13, “Environment Variables”.

Use of `MYSQL_PWD` is insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”.

4.2 Using MySQL Programs

4.2.1 Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. “`shell>`” represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for `sh` or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
shell> mysql --user=root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump -u root personnel
```

Arguments that begin with a single or double dash (“`-`”, “`--`”) specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode. Option syntax is described in Section 4.2.3, “Specifying Program Options”.

Nonoption arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first nonoption argument as a database name, so the command `mysql --user=root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program supports and describe the meaning of any additional nonoption arguments.

Some options are common to a number of programs. The most frequently used of these are the `--host` [226] (or `-h`), `--user` [227] (or `-u`), and `--password` [226] (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the user name and password of your MySQL account. All MySQL client programs understand these options; they enable you to specify which server to connect to and the account to use on that server. Other connection options are `--port` [226] (or `-P`) to specify a TCP/IP port number and `--socket` [227] (or `-S`) to specify a Unix socket file on Unix (or named pipe name on Windows). For more information on options that specify connection options, see Section 4.2.2, “Connecting to the MySQL Server”.

You may find it necessary to invoke MySQL programs using the path name to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the path name of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you can run the program by invoking it as `mysql`, and it is not necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in Section 4.2.4, “Setting Environment Variables”.) After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.2.2 Connecting to the MySQL Server

For a client program to be able to connect to the MySQL server, it must use the proper connection parameters, such as the name of the host where the server is running and the user name and password of your MySQL account. Each connection parameter has a default value, but you can override them as necessary using program options specified either on the command line or in an option file.

The examples here use the `mysql` client program, but the principles apply to other clients such as `mysqldump`, `mysqladmin`, or `mysqlshow`.

This command invokes `mysql` without specifying any connection parameters explicitly:

```
shell> mysql
```

Because there are no parameter options, the default values apply:

- The default host name is `localhost`. On Unix, this has a special meaning, as described later.
- The default user name is `ODBC` on Windows or your Unix login name on Unix.
- No password is sent if neither `-p` nor `--password` [226] is given.
- For `mysql`, the first nonoption argument is taken as the name of the default database. If there is no such option, `mysql` does not select a default database.

To specify the host name and user name explicitly, as well as a password, supply appropriate options on the command line:

```
shell> mysql --host=localhost --user=myname --password=mypass mydb
shell> mysql -h localhost -u myname -pmypass mydb
```

For password options, the password value is optional:

- If you use a `-p` or `--password` [226] option and specify the password value, there must be *no space* between `-p` or `--password=` [226] and the password following it.
- If you use a `-p` or `--password` [226] option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Other users on your system may be able to see a password specified on the command line by executing a command such as `ps auxw`. See Section 5.4.2.2, “End-User Guidelines for Password Security”.

As just mentioned, including the password value on the command line can be a security risk. To avoid this problem, specify the `--password` or `-p` option without any following password value:

```
shell> mysql --host=localhost --user=myname --password mydb
shell> mysql -h localhost -u myname -p mydb
```

When the password option has no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, `mydb` is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That is a problem with the system library, not with MySQL. Internally, MySQL does not have any limit for the length of the password. To work around the problem, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

On Unix, MySQL programs treat the host name `localhost` specially, in a way that is likely different from what you expect compared to other network-based programs. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file. This occurs even if a `--port` [226] or `-P` option is given to specify a port number. To ensure that the client makes a TCP/IP connection to the local server, use `--host` [226] or `-h` to specify a host name value of `127.0.0.1`, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for `localhost`, by using the `--protocol=TCP` [226] option. For example:

```
shell> mysql --host=127.0.0.1
shell> mysql --protocol=TCP
```

The `--protocol` [226] option enables you to establish a particular type of connection even when the other options would normally default to some other protocol.

On Windows, you can force a MySQL client to use a named-pipe connection by specifying the `--pipe` [226] or `--protocol=PIPE` [226] option, or by specifying `.` (period) as the host name. If named-pipe connections are not enabled, an error occurs. Use the `--socket` [227] option to specify the name of the pipe if you do not want to use the default pipe name.

Connections to remote servers always use TCP/IP. This command connects to the server running on `remote.example.com` using the default port number (3306):

```
shell> mysql --host=remote.example.com
```

To specify a port number explicitly, use the `--port` [226] or `-P` option:

```
shell> mysql --host=remote.example.com --port=13306
```

You can specify a port number for connections to a local server, too. However, as indicated previously, connections to `localhost` on Unix will use a socket file by default. You will need to force a TCP/IP connection as already described or any option that specifies a port number will be ignored.

For this command, the program uses a socket file on Unix and the `--port` [226] option is ignored:

```
shell> mysql --port=13306 --host=localhost
```

To cause the port number to be used, invoke the program in either of these ways:

```
```

```
shell> mysql --port=13306 --host=127.0.0.1
shell> mysql --port=13306 --protocol=TCP
```

The following list summarizes the options that can be used to control how client programs connect to the server:

- `--host=host_name` [226], `-h host_name`

  The host where the server is running. The default value is `localhost`.

- `--password[=pass_val]` [226], `-p[pass_val]`

  The password of the MySQL account. As described earlier, the password value is optional, but if given, there must be *no space* between `-p` or `--password=` [226] and the password following it. The default is to send no password.

- `--pipe` [226], `-W`

  On Windows, connect to the server using a named pipe. The server must be started with the `--enable-named-pipe` [386] option to enable named-pipe connections.

- `--port=port_num` [226], `-P port_num`

  The port number to use for the connection, for connections made using TCP/IP. The default port number is 3306.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [226]

  This option explicitly specifies a protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For example, connections on Unix to `localhost` are made using a Unix socket file by default:

  ```
  shell> mysql --host=localhost
  ```

  To force a TCP/IP connection to be used instead, specify a `--protocol` [226] option:

  ```
  shell> mysql --host=localhost --protocol=TCP
  ```

  The following table shows the permissible `--protocol` [226] option values and indicates the platforms on which each value may be used. The values are not case sensitive.

`--protocol` [226] Value	Connection Protocol	Permissible Operating Systems
`TCP`	TCP/IP connection to local or remote server	All
`SOCKET`	Unix socket file connection to local server	Unix only
`PIPE`	Named-pipe connection to local or remote server	Windows only
`MEMORY`	Shared-memory connection to local server	Windows only

  The `--protocol` [226] option was added in MySQL 4.1.

- `--shared-memory-base-name=name` [226]

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

The server must be started with the `--shared-memory` [392] option to enable shared-memory connections.

- `--socket=file_name` [227], `-S file_name`

  On Unix, the name of the Unix socket file to use, for connections made using a named pipe to a local server. The default Unix socket file name is `/tmp/mysql.sock`.

  On Windows, the name of the named pipe to use, for connections to a local server. The default Windows pipe name is `MySQL`. The pipe name is not case sensitive.

  The server must be started with the `--enable-named-pipe` [386] option to enable named-pipe connections.

- `--ssl*`

  Options that begin with `--ssl` [521] are used for establishing a secure connection to the server using SSL, if the server is configured with SSL support. For details, see Section 5.6.6.3, "SSL Command Options".

- `--user=user_name` [227], `-u user_name`

  The user name of the MySQL account you want to use. The default user name is `ODBC` on Windows or your Unix login name on Unix.

It is possible to specify different default values to be used when you make a connection so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the `[client]` section of an option file. The relevant section of the file might look like this:

```
[client]
host=host_name
user=user_name
password=your_pass
```

  Section 4.2.3.3, "Using Option Files", discusses option files further.

- You can specify some connection parameters using environment variables. The host can be specified for `mysql` using `MYSQL_HOST`. The MySQL user name can be specified using `USER` (this is for Windows and NetWare only). The password can be specified using `MYSQL_PWD`, although this is insecure; see Section 5.4.2.2, "End-User Guidelines for Password Security". For a list of variables, see Section 2.13, "Environment Variables".

4.2.3 Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables (see Section 4.2.4, "Setting Environment Variables"). This method is useful for options that you want to apply each time the program runs. In practice, option

files are used more commonly for this purpose, but Section 5.7.2, “Running Multiple Servers on Unix”, discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for the server and for client programs.

Options are processed in order, so if an option is specified multiple times, the last occurrence takes precedence. The following command causes `mysql` to connect to the server running on `localhost`:

```
shell> mysql -h example.com -h localhost
```

If conflicting or related options are given, later options take precedence over earlier options. The following command runs `mysql` in “no column names” mode:

```
shell> mysql --column-names --skip-column-names
```

MySQL programs determine which options are given first by examining environment variables, then by reading option files, and then by checking the command line. This means that environment variables have the lowest precedence and command-line options the highest.

You can take advantage of the way that MySQL programs process options by specifying default option values for a program in an option file. That enables you to avoid typing them each time you run the program while enabling you to override the defaults if necessary by using command-line options.

An option can be specified by writing it in full or as any unambiguous prefix. For example, the `--compress` [293] option can be given to `mysqldump` as `--compr`, but not as `--comp` because the latter is ambiguous:

```
shell> mysqldump --comp
mysqldump: ambiguous option '--comp' (compatible, compress)
```

Be aware that the use of option prefixes can cause problems in the event that new options are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

4.2.3.1 Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it is a short form or long form of the option name. Many options have both short and long forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` [226] indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an “`=`” sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=`*`pass_val`* [226] or as `--password` [226]. In the latter case (with no password value

given), the program prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

```
shell> mysql -ptest
shell> mysql -p test
```

The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.

- Within option names, dash ("`-`") and underscore ("`_`") may be used interchangeably. For example, `--skip-grant-tables` [392] and `--skip_grant_tables` [392] are equivalent. (However, the leading dashes cannot be given as underscores.)

- For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. For example, the following command tells `mysqladmin` to ping the server 1024 times, sleeping 10 seconds between each ping:

```
mysql> mysqladmin --count=1K --sleep=10 ping
```

Option values that contain spaces must be quoted when given on the command line. For example, the `--execute` [261] (or `-e`) option can be used with `mysql` to pass SQL statements to the server. When this option is used, `mysql` executes the statements in the option value and exits. The statements must be enclosed by quotation marks. For example, you can use the following command to obtain a list of user accounts:

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
Enter password: ******
+------+-----------+
| User | Host      |
+------+-----------+
|      | gigan     |
| root | gigan     |
|      | localhost |
| jon  | localhost |
| root | localhost |
+------+-----------+
shell>
```

Note that the long form (`--execute` [261]) is followed by an equal sign (`=`).

If you wish to use quoted values within a statement, you will either need to escape the inner quotation marks, or use a different type of quotation marks within the statement from those used to quote the statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotation marks, you can use double quotation marks around the statement, and single quotation marks for any quoted values within the statement.

Multiple SQL statements may be passed in the option value on the command line, separated by semicolons:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: ******
```

```
+------------+
| VERSION()  |
+------------+
| 4.1.17-log |
+------------+
+---------------------+
| NOW()               |
+---------------------+
| 2006-01-05 21:19:04 |
+---------------------+
```

The `--execute` or `-e` option may also be used to pass commands in an analogous fashion to the `ndb_mgm` management client for MySQL Cluster. See Section 15.2.5, “Safe Shutdown and Restart of MySQL Cluster”, for an example.

4.2.3.2 Program Option Modifiers

MySQL 4.0.2 introduced some additional flexibility in the way you specify options. SQL 4.0.2. Some of these changes relate to the way you specify options that have “enabled” and “disabled” states, and to the use of options that might be present in one version of MySQL but not another. Those capabilities are discussed in this section.

Some options are “boolean” and control behavior that can be turned on or off. Some options control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` [260] option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

Another change to option processing introduced in MySQL 4.0.2 is that you can use the `--loose` prefix for command-line options. If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file, An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it. This strategy requires that all versions involved be 4.0.2 or later, because earlier versions know nothing of the `--loose` convention.

As of MySQL 4.0.2, `mysqld` enables a limit to be placed on how large client programs can set dynamic system variables. To do this, use a `--maximum` prefix with the variable name. For example, `--maximum-query_cache_size=4M` prevents any client from making the query cache size larger than 4MB.

4.2.3.3 Using Option Files

Most MySQL programs can read startup options from option files (also sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program. Option file capability is available from MySQL 3.22 on. For the MySQL server, MySQL provides a number of preconfigured option files.

To determine whether a program reads option files, invoke it with the `--help` option. (For `mysqld`, use `--verbose` [395] and `--help` [384] as of MySQL 4.1.1.) If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note

Option files used with MySQL Cluster programs are covered in Section 15.3, "MySQL Cluster Configuration".

On Windows, MySQL programs read startup options from the following files.

File Name	Purpose
`%WINDIR%\my.ini`, `%WINDIR%\my.cnf`	Global options
`C:\my.ini`, `C:\my.cnf`	Global options
`INSTALLDIR``\my.ini`, *`INSTALLDIR`*`\my.cnf`	Global options
`defaults-extra-file`	The file specified with `--defaults-extra-file=`*`path`* [235], if any

Note

Programs look for option files using both extensions (`.ini`, `.cnf`) in all locations only as of MySQL 4.0.23 and 4.1.8. Before MySQL 4.0.23 and 4.1.8, programs look in *`WINDIR`* and *`INSTALLDIR`* only for `my.ini`, and in `C:\` only for `my.cnf`.

`%WINDIR%` represents the location of your Windows directory. This is commonly `C:\WINDOWS` or `C:\WINNT`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

`INSTALLDIR` represents the MySQL installation directory. With MySQL 4.1.5 and up, this is typically `C:\`*`PROGRAMDIR`*`\MySQL\MySQL 4.1 Server` where *`PROGRAMDIR`* represents the programs directory (usually `Program Files` on English-language versions of Windows), when MySQL 4.1 has been installed using the installation and configuration wizards. See Section 2.3.4.14, "The Location of the my.ini File".

On Unix, MySQL programs read startup options from the following files.

File Name	Purpose
`/etc/my.cnf`	Global options
`DATADIR/my.cnf`	Server-specific options
`defaults-extra-file`	The file specified with `--defaults-extra-file=`*`path`* [235], if any

File Name	Purpose
`~/.my.cnf`	User-specific options

`~` represents the current user's home directory (the value of `$HOME`).

`DATADIR` represents the path to the directory in which the server-specific `my.cnf` file resides.

Typically, *`DATADIR`* is `/usr/local/mysql/data` for a binary installation or `/usr/local/var` for a source installation. Note that this is the data directory location that was specified at configuration time, not the one specified with the `--datadir` [385] option when `mysqld` starts. Use of `--datadir` [385] at runtime has no effect on where the server looks for option files, because it looks for them before processing any options.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor.

If multiple instances of a given option are found, the last instance takes precedence. There is one exception: For `mysqld`, the *first* instance of the `--user` [395] option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional as a security measure.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option.

The syntax for specifying options in an option file is similar to command-line syntax (see Section 4.2.3.1, “Using Options on the Command Line”). However, in an option file, you omit the leading two dashes from the option name and you specify only one option per line. For example, `--quick` and `--host=localhost` on the command line should be specified as `quick` and `host=localhost` on separate lines in an option file. To specify an option of the form `--loose-`*`opt_name`* in an option file, write it as `loose-`*`opt_name`*.

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- *`#comment`*, *`;comment`*

  Comment lines start with “`#`” or “`;`”. As of MySQL 4.0.14, a “`#`” comment can start in the middle of a line as well.

- `[`*`group`*`]`

  `group` is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given.

- *`opt_name`*

  This is equivalent to `--`*`opt_name`* on the command line.

- *`opt_name=value`*

  This is equivalent to `--`*`opt_name=value`* on the command line. In an option file, you can have spaces around the “`=`” character, something that is not true on the command line. As of MySQL 4.0.16, you can

optionally enclose the value within double quotation marks or single quotation marks. This is useful if the value contains a "#" comment character.

- `set-variable = var_name=value`

  Set the program variable *var_name* to the given value. This is equivalent to `--set-variable=var_name=value` on the command line. Spaces are permitted around the first "=" character but not around the second. This syntax is deprecated as of MySQL 4.0. See Section 4.2.3.4, "Using Options to Set Program Variables", for more information on setting program variables.

Leading and trailing spaces are automatically deleted from option names and values.

You can use the escape sequences "\b", "\t", "\n", "\r", "\\", and "\s" in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters. The escaping rules in option files are:

- If a backslash is followed by a valid escape sequence character, the sequence is converted to the character represented by the sequence. For example, "\s" is converted to a space.
- If a backslash is not followed by a valid escape sequence character, it remains unchanged. For example, "\S" is retained as is.

The preceding rules mean that a literal backslash can be given as "\\", or as "\" if it is not followed by a valid escape sequence character.

The rules for escape sequences in option files differ slightly from the rules for escape sequences in string literals in SQL statements. In the latter context, if "*x*" is not a value escape sequence character, "*x*" becomes "*x*" rather than "*x*". See Section 8.1.1, "String Literals".

The escaping rules for option file values are especially pertinent for Windows path names, which use "\" as a path name separator. A separator in a Windows path name must be written as "\\" if it is followed by an escape sequence character. It can be written as "\\" or "\" if it is not. Alternatively, "/" may be used in Windows path names and will be treated as "\". Suppose that you want to specify a base directory of `C:\Program Files\MySQL\MySQL Server 4.1` in an option file. This can be done several ways. Some examples:

```
basedir="C:\Program Files\MySQL\MySQL Server 4.1"
basedir="C:\\Program Files\\MySQL\\MySQL Server 4.1"
basedir="C:/Program Files/MySQL/MySQL Server 4.1"
basedir=C:\\Program\sFiles\\MySQL\\MySQL\sServer\s4.1
```

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the `mysqld` server and the `mysql` client program, respectively.

The `[client]` option group is read by all client programs (but *not* by `mysqld`). This enables you to specify options that apply to all clients. For example, `[client]` is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure not to put an option in the `[client]` group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock
```

```
[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

The preceding option file uses *`var_name=value`* syntax for the lines that set the `key_buffer_size` [415] and `max_allowed_packet` [418] variables. Prior to MySQL 4.0.2, you must use `set-variable` syntax instead (described earlier in this section).

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
set-variable = connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

This option file uses `set-variable` syntax to set the `connect_timeout` [409] variable. For MySQL 4.0.2 and up, you can also set the variable using just `connect_timeout = 2` [409].

As of MySQL 4.0.14, if you want to create option groups that should be read only by `mysqld` servers from a specific MySQL release series only, you can do this by using groups with names of `[mysqld-4.0]`, `[mysqld-4.1]`, and so forth. The following group indicates that the `--new` option should be used only by MySQL servers with 4.0.x version numbers:

```
[mysqld-4.0]
new
```

Beginning with MySQL 4.1.11, it is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

There is no guarantee about the order in which the option files in the directory will be read.

Note

Currently, any files to be found and included using the `!includedir` directive on Unix operating systems *must* have file names ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqldamin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

Command-Line Options that Affect Option-File Handling

Most MySQL programs that support option files handle the following options. They affect option-file handling, so they must be given on the command line and not in an option file. To work properly, each of these options must immediately follow the command name, with these exceptions:

- `--print-defaults` [235] may be used immediately after `--defaults-file` [235] or `--defaults-extra-file` [235].
- On Windows, if the `--defaults-file` [235] and `--install` options are given, `--install` option must be first. See Section 2.3.11, “Starting MySQL as a Windows Service”.

When specifying file names, you should avoid the use of the “~” shell metacharacter because it might not be interpreted as you expect.

- `--defaults-extra-file=`*`file_name`* [235]

  Read this option file after the global option file but (on Unix) before the user option file. *`file_name`* is the full path name to the file.

- `--defaults-file=`*`file_name`* [235]

  Use only the given option file. *`file_name`* is the full path name to the file. If the file does not exist, the program exits with an error.

- `--no-defaults` [235]

  Do not read any option files. If a program does not start because it is reading unknown options from an option file, `--no-defaults` [235] can be used to prevent the program from reading them.

- `--print-defaults` [235]

  Print the program name and all options that it gets from option files.

Preconfigured Option Files

MySQL provides a number of preconfigured option files that can be used as a basis for tuning the MySQL server. Look for files such as `my-small.cnf`, `my-medium.cnf`, `my-large.cnf`, and `my-huge.cnf`,

which are sample option files for small, medium, large, and very large systems. On Windows, the extension is `.ini` rather than `.cnf`.

Note

On Windows, the `.ini` or `.cnf` option file extension might not be displayed.

For a binary distribution, look for the files in or under your installation directory. If you have a source distribution, look in the `support-files` directory. You can rename a copy of a sample file and place it in the appropriate location for use as a base configuration file. Regarding names and appropriate location, see the general information provided in Section 4.2.3.3, “Using Option Files”.

4.2.3.4 Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime using the `SET` statement. See Section 12.4.4, “`SET` Syntax”, and Section 5.1.4, “Using System Variables”.

As of MySQL 4.0.2, most of these program variables also can be set at server startup by using the same syntax that applies to specifying program options. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M

[mysqld]
key-buffer-size=512M
```

A variable can be specified by writing it in full or as any unambiguous prefix. For example, the `max_allowed_packet` variable can be set for `mysql` as `--max_a`, but not as `--max` because the latter is ambiguous:

```
```

```
shell> mysql --max=1000000
mysql: ambiguous option '--max=1000000' (max_allowed_packet, max_join_size)
```

Be aware that the use of variable prefixes can cause problems in the event that new variables are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

Prior to MySQL 4.0.2, program variable names are not recognized as option names. Instead, use the `--set-variable` option to assign a value to a variable:

```
shell> mysql --set-variable=max_allowed_packet=16777216
shell> mysql --set-variable=max_allowed_packet=16M
```

In an option file, omit the leading dashes:

```
[mysql]
set-variable = max_allowed_packet=16777216
```

Or:

```
[mysql]
set-variable = max_allowed_packet=16M
```

With `--set-variable`, underscores in variable names cannot be given as dashes for versions of MySQL older than 4.0.2, and the variable name must be specified in full.

The `--set-variable` option is still recognized in MySQL 4.0.2 and up, but is deprecated.

4.2.3.5 Option Defaults, Options Expecting Values, and the = Sign

By convention, long forms of options that assign a value are written with an equals (`=`) sign, like this:

```
shell> mysql --host=tonfisk --user=jon
```

For options that require a value (that is, not having a default value), the equal sign is not required, and so the following is also valid:

```
shell> mysql --host tonfisk --user jon
```

In both cases, the `mysql` client attempts to connect to a MySQL server running on the host named “tonfisk” using an account with the user name “jon”.

Due to this behavior, problems can occasionally arise when no value is provided for an option that expects one. Consider the following example, where a user connects to a MySQL server running on host `tonfisk` as user `jon`:

```
shell> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 4.1.25 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| jon@%          |
+----------------+
1 row in set (0.00 sec)
```

Omitting the required value for one of these option yields an error, such as the one shown here:

```
shell> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

In this case, `mysql` was unable to find a value following the `--user` [227] option because nothing came after it on the command line. However, if you omit the value for an option that is *not* the last option to be used, you obtain a different error that you may not be expecting:

```
shell> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

Because `mysql` assumes that any string following `--host` [226] on the command line is a host name, `--host` [226] `--user` [227] is interpreted as `--host=--user` [226], and the client attempts to connect to a MySQL server running on a host named “--user”.

Options having default values always require an equal sign when assigning a value; failing to do so causes an error. For example, the MySQL server `--log-error` [388] option has the default value *`host_name`*`.err`, where *`host_name`* is the name of the host on which MySQL is running. Assume that you are running MySQL on a computer whose host name is “tonfisk”, and consider the following invocation of `mysqld_safe`:

```
shell> mysqld_safe &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

After shutting down the server, restart it as follows:

```
shell> mysqld_safe --log-error &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

The result is the same, since `--log-error` is not followed by anything else on the command line, and it supplies its own default value. (The `&` character tells the operating system to run MySQL in the

background; it is ignored by MySQL itself.) Now suppose that you wish to log errors to a file named `my-errors.err`. You might try starting the server with `--log-error my-errors`, but this does not have the intended effect, as shown here:

```
shell> mysqld_safe --log-error my-errors &
[1] 31357
shell> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+  Done                    ./mysqld_safe --log-error my-errors
```

The server attempted to start using `/usr/local/mysql/var/tonfisk.err` as the error log, but then shut down. Examining the last few lines of this file shows the reason:

```
shell> tail /usr/local/mysql/var/tonfisk.err
080111 22:53:32  InnoDB: Started; log sequence number 0 46409
/usr/local/mysql/libexec/mysqld: Too many arguments (first extra is 'my-errors').
Use --verbose --help to get a list of available options
080111 22:53:32 [ERROR] Aborting

080111 22:53:32  InnoDB: Starting shutdown...
080111 22:53:34  InnoDB: Shutdown completed; log sequence number 0 46409
080111 22:53:34 [Note] /usr/local/mysql/libexec/mysqld: Shutdown complete

080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended
```

Because the `--log-error` option supplies a default value, you must use an equal sign to assign a different value to it, as shown here:

```
shell> mysqld_safe --log-error=my-errors &
[1] 31437
shell> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

shell>
```

Now the server has been started successfully, and is logging errors to the file `/usr/local/mysql/var/my-errors.err`.

Similar issues can arise when specifying option values in option files. For example, consider a `my.cnf` file that contains the following:

```
[mysql]

host
user
```

When the `mysql` client reads this file, these entries are parsed as `--host` [261] `--user` [265] or `--host=--user` [261], with the result shown here:

```
shell> mysql
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

However, in option files, an equal sign is not assumed. Suppose the `my.cnf` file is as shown here:

```
[mysql]
```

```
user jon
```

Trying to start `mysql` in this case causes a different error:

```
shell> mysql
mysql: unknown option '--user jon'
```

A similar error would occur if you were to write `host tonfisk` in the option file rather than `host=tonfisk`. Instead, you must use the equal sign:

```
[mysql]

user=jon
```

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 4.1.25 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
1 row in set (0.00 sec)
```

This is not the same behavior as with the command line, where the equal sign is not required:

```
shell> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 4.1.25 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@tonfisk   |
+---------------+
1 row in set (0.00 sec)
```

4.2.4 Setting Environment Variables

Environment variables can be set at the command prompt to affect the current invocation of your command processor, or set permanently to affect future invocations. To set a variable permanently, you can set it in a startup file or by using the interface provided by your system for this purpose. Consult the documentation for your command interpreter for specific details. Section 2.13, “Environment Variables”, lists all environment variables that affect MySQL program operation.

To specify a value for an environment variable, use the syntax appropriate for your command processor. For example, on Windows or NetWare, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For `csh` and `tcsh`, use `setenv` to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, use the interface provided by your system or place the appropriate command or commands in a startup file that your command interpreter reads each time it starts.

On Windows, you can set environment variables using the System Control Panel (under Advanced).

On Unix, typical shell startup files are `.bashrc` or `.bash_profile` for `bash`, or `.tcshrc` for `tcsh`.

Suppose that your MySQL programs are installed in `/usr/local/mysql/bin` and that you want to make it easy to invoke these programs. To do this, set the value of the `PATH` environment variable to include that directory. For example, if your shell is `bash`, add the following line to your `.bashrc` file:

```
PATH=${PATH}:/usr/local/mysql/bin
```

`bash` uses different startup files for login and nonlogin shells, so you might want to add the setting to `.bashrc` for login shells and to `.bash_profile` for nonlogin shells to make sure that `PATH` is set regardless.

If your shell is `tcsh`, add the following line to your `.tcshrc` file:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If the appropriate startup file does not exist in your home directory, create it with a text editor.

After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.3 MySQL Server and Server-Startup Programs

This section describes `mysqld`, the MySQL server, and several programs that are used to start the server.

4.3.1 `mysqld` — The MySQL Server

`mysqld`, also known as MySQL Server, is the main program that does most of the work in a MySQL installation. MySQL Server manages access to the MySQL data directory that contains databases and tables. The data directory is also the default location for other information such as log files and status files.

When MySQL server starts, it listens for network connections from client programs and manages access to databases on behalf of those clients.

The mysqld program has many options that can be specified at startup. For a complete list of options, run this command:

```
shell> mysqld --verbose --help
```

For versions older than MySQL 4.1.1, leave out the --verbose [395] option.

MySQL Server also has a set of system variables that affect its operation as it runs. System variables can be set at server startup, and many of them can be changed at runtime to effect dynamic server reconfiguration. MySQL Server also has a set of status variables that provide information about its operation. You can monitor these status variables to access runtime performance characteristics.

For a full description of MySQL Server command options, system variables, and status variables, see Section 5.1, “The MySQL Server”. For information about installing MySQL and setting up the initial configuration, see Chapter 2, *Installing and Upgrading MySQL*.

4.3.2 mysqld_safe — MySQL Server Startup Script

mysqld_safe is the recommended way to start a mysqld server on Unix and NetWare. mysqld_safe adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log file. NetWare-specific behaviors are listed later in this section.

Note

Before MySQL 4.0, mysqld_safe is named safe_mysqld. To preserve backward compatibility, MySQL binary distributions include safe_mysqld as a symbolic link to mysqld_safe until MySQL 5.1.

By default, mysqld_safe tries to start an executable named mysqld-max if it exists, and mysqld otherwise. Be aware of the implications of this behavior:

- On Linux, the MySQL-Max RPM relies on this mysqld_safe behavior. The RPM installs an executable named mysqld-max, which causes mysqld_safe to automatically use that executable rather than mysqld from that point on.
- If you install a MySQL-Max distribution that includes a server named mysqld-max, and then upgrade later to a non-Max version of MySQL, mysqld_safe will still attempt to run the old mysqld-max server. If you perform such an upgrade, you should manually remove the old mysqld-max server to ensure that mysqld_safe runs the new mysqld server.

To override the default behavior and specify explicitly the name of the server you want to run, specify a --mysqld [244] or --mysqld-version [244] option to mysqld_safe. You can also use --ledir [244] to indicate the directory where mysqld_safe should look for the server.

Many of the options to mysqld_safe are the same as the options to mysqld. See Section 5.1.2, “Server Command Options”.

Options unknown to mysqld_safe are passed to mysqld if they are specified on the command line, but ignored if they are specified in the [mysqld_safe] group of an option file. See Section 4.2.3.3, “Using Option Files”.

mysqld_safe reads all options from the [mysqld], [server], and [mysqld_safe] sections in option files. For example, if you specify a [mysqld] section like this, mysqld_safe will find and use the --log-error option:

```
[mysqld]
```

```
log-error=error.log
```

For backward compatibility, mysqld_safe also reads [safe_mysqld] sections, although you should rename such sections to [mysqld_safe] when you begin using MySQL 4.0 or later.

mysqld_safe supports the following options. It also reads option files and supports the options for processing them described at Command-Line Options that Affect Option-File Handling.

Table 4.1 mysqld_safe Options

Format	Option File	Description
--autoclose [243]	autoclose [243]	On NetWare, mysqld_safe provides a screen presence
--basedir=path [244]	basedir [244]	The path to the MySQL installation directory
--core-file-size=size [244]	core-file-size [244]	The size of the core file that mysqld should be able to create
--datadir=path [244]	datadir [244]	The path to the data directory
--defaults-extra-file=path [244]	defaults-extra-file [244]	The name of an option file to be read in addition to the usual option files
--defaults-file=file_name [244]	defaults-file [244]	The name of an option file to be read instead of the usual option files
--help		Display a help message and exit
--ledir=path [244]	ledir [244]	Use this option to indicate the path name to the directory where the server is located
--log-error=file_name	log-error	Write the error log to the given file
--mysqld=prog_name [244]	mysqld [244]	The name of the server program (in the ledir directory) that you want to start
--mysqld-version=suffix [244]	mysqld-version [244]	This option is similar to the --mysqld option, but you specify only the suffix for the server program name
--nice=priority [245]	nice [245]	Use the nice program to set the server's scheduling priority to the given value
--no-defaults [245]	no-defaults [245]	Do not read any option files
--open-files-limit=count [245]	open-files-limit [245]	The number of files that mysqld should be able to open
--pid-file=file_name [245]	pid-file=file_name [245]	The path name of the process ID file
--port=number [245]	port [245]	The port number that the server should use when listening for TCP/IP connections
--skip-kill-mysqld [245]	skip-kill-mysqld [245]	Do not try to kill stray mysqld processes
--socket=path [245]	socket [245]	The Unix socket file that the server should use when listening for local connections
--timezone=timezone [245]	timezone [245]	Set the TZ time zone environment variable to the given option value
--user={user_name|user_id} [245]	user [245]	Run the mysqld server as the user having the name user_name or the numeric user ID user_id

- --autoclose [243]

(NetWare only) On NetWare, mysqld_safe provides a screen presence. When you unload (shut down) the mysqld_safe NLM, the screen does not by default go away. Instead, it prompts for user input:

```
*<NLM has terminated; Press any key to close the screen>*
```

If you want NetWare to close the screen automatically instead, use the --autoclose [243] option to mysqld_safe.

- --basedir=*path* [244]

  The path to the MySQL installation directory.

- --core-file-size=*size* [244]

  The size of the core file that mysqld should be able to create. The option value is passed to ulimit -c.

- --datadir=*path* [244]

  The path to the data directory.

- --defaults-extra-file=*path* [244]

  The name of an option file to be read in addition to the usual option files. This must be the first option on the command line if it is used.

- --defaults-file=*file_name* [244]

  The name of an option file to be read instead of the usual option files. This must be the first option on the command line if it is used.

- --err-log=*file_name* [244]

  The old form of the --log-error option, to be used before MySQL 4.0.

- --ledir=*path* [244]

  If mysqld_safe cannot find the server, use this option to indicate the path name to the directory where the server is located.

- --log-error=*file_name*

  Write the error log to the given file. See Section 5.3.1, “The Error Log”.

- --mysqld=*prog_name* [244]

  The name of the server program (in the ledir directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If mysqld_safe cannot find the server, use the --ledir [244] option to indicate the path name to the directory where the server is located.

- --mysqld-version=*suffix* [244]

  This option is similar to the --mysqld [244] option, but you specify only the suffix for the server program name. The basename is assumed to be mysqld. For example, if you use --mysqld-version=max [244], mysqld_safe starts the mysqld-max program in the ledir directory. If the argument to --mysqld-version [244] is empty, mysqld_safe uses mysqld in the ledir directory.

- `--nice=priority` [245]

  Use the `nice` program to set the server's scheduling priority to the given value. This option was added in MySQL 4.0.14.

- `--no-defaults` [245]

  Do not read any option files. This must be the first option on the command line if it is used.

- `--open-files-limit=count` [245]

  The number of files that `mysqld` should be able to open. The option value is passed to `ulimit -n`. Note that you need to start `mysqld_safe` as `root` for this to work properly.

- `--pid-file=file_name` [245]

  The path name of the process ID file.

- `--port=port_num` [245]

  The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

- `--skip-kill-mysqld` [245]

  Do not try to kill stray `mysqld` processes at startup. This option works only on Linux.

- `--socket=path` [245]

  The Unix socket file that the server should use when listening for local connections.

- `--timezone=timezone` [245]

  Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={user_name|user_id}` [245]

  Run the `mysqld` server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute `mysqld_safe` with the `--defaults-file` [244] or `--defaults-extra-file` [244] option to name an option file, the option must be the first one given on the command line or the option file will not be used. For example, this command will not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The `mysqld_safe` script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See Section 2.1.5, “Installation Layouts”.) `mysqld_safe` expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which `mysqld_safe` is invoked). For binary distributions, `mysqld_safe` looks under its working directory

for bin and data directories. For source distributions, it looks for libexec and var directories. This condition should be met if you execute mysqld_safe from your MySQL installation directory (for example, /usr/local/mysql for a binary distribution).

- If the server and databases cannot be found relative to the working directory, mysqld_safe attempts to locate them by absolute path names. Typical locations are /usr/local/libexec and /usr/local/var. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because mysqld_safe tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run mysqld_safe from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If mysqld_safe fails, even when invoked from the MySQL installation directory, you can specify the --ledir [244] and --datadir [244] options to indicate the directories in which the server and databases are located on your system.

Normally, you should not edit the mysqld_safe script. Instead, configure mysqld_safe by using command-line options or options in the [mysqld_safe] section of a my.cnf option file. In rare cases, it might be necessary to edit mysqld_safe to get it to start the server properly. However, if you do this, your modified version of mysqld_safe might be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.

On NetWare, mysqld_safe is a NetWare Loadable Module (NLM) that is ported from the original Unix shell script. It starts the server as follows:

1. Runs a number of system and option checks.
2. Runs a check on MyISAM and ISAM tables.
3. Provides a screen presence for the MySQL server.
4. Starts mysqld, monitors it, and restarts it if it terminates in error.
5. Sends error messages from mysqld to the *host_name*.err file in the data directory.
6. Sends mysqld_safe screen output to the *host_name*.safe file in the data directory.

4.3.3 mysql.server — MySQL Server Startup Script

MySQL distributions on Unix include a script named mysql.server. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the Mac OS X Startup Item for MySQL.

mysql.server can be found in the support-files directory under your MySQL installation directory or in a MySQL source distribution.

If you use the Linux server RPM package (MySQL-server-*VERSION*.rpm), the mysql.server script will be installed in the /etc/init.d directory with the name mysql. You need not install it manually. See Section 2.4, “Installing MySQL from RPM Packages on Linux”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as mysqld.

If you install MySQL from a source distribution or using a binary distribution format that does not install mysql.server automatically, you can install it manually. Instructions are provided in Section 2.10.2.2, “Starting and Stopping MySQL Automatically”.

mysql.server reads options from the [mysql.server] and [mysqld] sections of option files. For backward compatibility, it also reads [mysql_server] sections, although you should rename such sections to [mysql.server] when you begin using MySQL 4.0 or later.

mysql.server supports the following options.

- --basedir=*path* [247]

  The path to the MySQL installation directory.

- --datadir=*path* [247]

  The path to the MySQL data directory.

- --pid-file=*file_name* [247]

  The path name of the file in which the server should write its process ID.

4.3.4 mysqld_multi — Manage Multiple MySQL Servers

mysqld_multi is designed to manage several mysqld processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status.

mysqld_multi searches for groups named [mysqld*N*] in my.cnf (or in the file named by the --config-file [248] option). *N* can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to mysqld_multi to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the [mysqld] group used for starting mysqld. (See, for example, Section 2.10.2.2, “Starting and Stopping MySQL Automatically”.) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see Section 5.7, “Running Multiple MySQL Servers on the Same Machine”.

To invoke mysqld_multi, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

start, stop, and report indicate which operation to perform. You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, mysqld_multi performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named [mysqld17] is 17. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value 10-13 represents groups [mysqld10] through [mysqld13]. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group [mysqld17]:

```
```

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups [mysqld8] and [mysqld10] through [mysqld13]:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

Option files read are searched for [mysqld_multi] and [mysqldN] option groups. The [mysqld_multi] group can be used for options to mysqld_multi itself. [mysqldN] groups can be used for options passed to specific mysqld instances.

mysqld_multi supports the following options.

- --help [248]

  Display a help message and exit.

- --config-file=*file_name* [248]

  Specify the name of an alternative option file. This affects where mysqld_multi looks for [mysqldN] option groups. Without this option, all options are read from the usual my.cnf file. The option does not affect where mysqld_multi reads its own options, which are always taken from the [mysqld_multi] group in the usual my.cnf file.

- --example [248]

  Display a sample option file.

- --log=*file_name* [248]

  Specify the name of the log file. If the file exists, log output is appended to it.

- --mysqladmin=*prog_name* [248]

  The mysqladmin binary to be used to stop servers.

- --mysqld=*prog_name* [248]

  The mysqld binary to be used. Note that you can specify mysqld_safe as the value for this option also. If you use mysqld_safe to start the server, you can include the mysqld or ledir options in the corresponding [mysqldN] option group. These options indicate the name of the server that mysqld_safe should start and the path name of the directory where the server is located. (See the descriptions for these options in Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.) Example:

  ```
  [mysqld38]
  mysqld = mysqld-max
  ledir  = /opt/local/mysql/libexec
  ```

- --no-log [248]

  Print log information to stdout rather than to the log file. By default, output goes to the log file.

- `--password=password` [249]

  The password of the MySQL account to use when invoking `mysqladmin`. Note that the password value is not optional for this option, unlike for other MySQL programs.

- `--silent` [249]

  Silent mode; disable warnings. This option was added in MySQL 4.1.6.

- `--tcp-ip` [249]

  Connect to each MySQL server through the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only through the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects `stop` and `report` operations.

- `--user=user_name`

  The user name of the MySQL account to use when invoking `mysqladmin`.

- `--verbose` [249]

  Be more verbose. This option was added in MySQL 4.1.6.

- `--version` [249]

  Display version information and exit.

Some notes about `mysqld_multi`:

- **Most important**: Before using `mysqld_multi` be sure that you understand the meanings of the options that are passed to the `mysqld` servers and *why* you would want to have separate `mysqld` processes. Beware of the dangers of using multiple `mysqld` servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See Section 5.7, “Running Multiple MySQL Servers on the Same Machine”.

- 

  Important

  Make sure that the data directory for each server is fully accessible to the Unix account that the specific `mysqld` process is started as. *Do not* use the Unix `root` account for this, unless you *know* what you are doing. See Section 5.4.6, “How to Run MySQL as a Normal User”.

- Make sure that the MySQL account used for stopping the `mysqld` servers (with the `mysqladmin` program) has the same user name and password for each server. Also, make sure that the account has the `SHUTDOWN` [492] privilege. If the servers that you want to manage have different user names or passwords for the administrative accounts, you might want to create an account on each server that has the same user name and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

  ```
  shell> mysql -u root -S /tmp/mysql.sock -p
  Enter password:
  mysql> GRANT SHUTDOWN ON *.*
      -> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
  ```

  See Section 5.5, “The MySQL Access Privilege System”. You have to do this for each `mysqld` server. Change the connection parameters appropriately when connecting to each one. Note that the host name

part of the account name must permit you to connect as multi_admin from the host where you want to run mysqld_multi.

- The Unix socket file and the TCP/IP port number must be different for every mysqld. (Alternatively, if the host has multiple network addresses, you can use --bind-address [384] to cause different servers to listen to different interfaces.)
- The --pid-file [245] option is very important if you are using mysqld_safe to start mysqld (for example, --mysqld=mysqld_safe [244]) Every mysqld should have its own process ID file. The advantage of using mysqld_safe instead of mysqld is that mysqld_safe monitors its mysqld process and restarts it if the process terminates due to a signal sent using kill -9 or for other reasons, such as a segmentation fault. Please note that the mysqld_safe script might require that you start it from a certain place. This means that you might have to change location to a certain directory before running mysqld_multi. If you have problems starting, please see the mysqld_safe script. Check especially the lines:

```
----------------------------------------------------------------
MY_PWD=`pwd`
# Check if we are starting this relative (for the binary release)
if test -d $MY_PWD/data/mysql -a -f ./share/mysql/english/errmsg.sys -a \
 -x ./bin/mysqld
----------------------------------------------------------------
```

  The test performed by these lines should be successful, or you might encounter problems. See Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.

- You might want to use the --user [395] option for mysqld, but to do this you need to run the mysqld_multi script as the Unix superuser (root). Having the option in the option file does not matter; you merely get a warning if you are not the superuser and the mysqld processes are started under your own Unix account.

The following example shows how you might set up an option file for use with mysqld_multi. The order in which the mysqld programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth [mysqldN] groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld     = /usr/local/bin/mysqld_safe
mysqladmin = /usr/local/bin/mysqladmin
user       = multi_admin
password   = multipass

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/var2/hostname.pid2
datadir    = /usr/local/mysql/var2
language   = /usr/local/share/mysql/english
user       = john

[mysqld3]
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/var3/hostname.pid3
datadir    = /usr/local/mysql/var3
```

```
language   = /usr/local/share/mysql/swedish
user       = monty

[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/var4/hostname.pid4
datadir    = /usr/local/mysql/var4
language   = /usr/local/share/mysql/estonia
user       = tonu

[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/var6/hostname.pid6
datadir    = /usr/local/mysql/var6
language   = /usr/local/share/mysql/japanese
user       = jani
```

See Section 4.2.3.3, “Using Option Files”.

4.4 MySQL Installation-Related Programs

The programs in this section are used when installing or upgrading MySQL.

4.4.1 `comp_err` — Compile MySQL Error Message File

`comp_err` creates the `errmsg.sys` file that is used by `mysqld` to determine the error messages to display for different error codes. `comp_err` normally is run automatically when MySQL is built. It compiles the `errmsg.sys` file from the plaintext file or files located in the `sql/share/language` directories in MySQL source distributions.

For more information about how error messages are defined, see the MySQL Internals Manual.

Invoke `comp_err` like this:

```
shell> comp_err [options] from_file ... to_file
```

The *`from_file`* arguments are the input files. *`to_file`* is the name of the output file.

`comp_err` supports the following options.

- `-?`, `-I`

  Display a help message and exit.

- `-# debug_options`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:O,file_name'`.

- `-V`

  Display version information and exit.

4.4.2 `make_win_src_distribution` — Create Source Distribution for Windows

`make_win_src_distribution` creates a Windows source package to be used on Windows systems. It is used after you configure and build the source distribution on a Unix or Unix-like system so that you have

a server binary to work with. (See the instructions at Section 2.9.7.2, “Creating a Windows Source Package from the Latest Development Source”.)

Invoke `make_win_src_distribution` like this from the top-level directory of a MySQL source distribution:

```
shell> make_win_src_distribution [options]
```

`make_win_src_distribution` understands the following options:

- `--help` [252]

  Display a help message and exit.

- `--debug` [252]

  Print information about script operations; do not create a package.

- `--dirname` [252]

  Directory name to copy files (intermediate).

- `--silent` [252]

  Do not print verbose list of files processed.

- `--suffix` [252]

  The suffix name for the package.

- `--tar` [252]

  Create a `.tar.gz` package instead of a `.zip` package.

  By default, `make_win_src_distribution` creates a Zip-format archive with the name `mysql-VERSION-win-src.zip`, where *`VERSION`* represents the version of your MySQL source tree.

- `--tmp` [252]

  Specify the temporary location.

4.4.3 `mysql_create_system_tables` — Generate Statements to Initialize MySQL System Tables

`mysql_create_system_tables` is a helper script that is invoked by `mysql_install_db` to generate the SQL statements required to initialize any grant tables that do not exist.

Invoke `mysql_create_system_tables` like this:

```
shell> mysql_create_system_tables {test|verbose} path_to_mysql_database host_name windows_option
```

The first argument is `test` (create entries for the `test` database) or `verbose` (display more information while the script runs. The second argument is the path to the `mysql` database directory. The third argument is the host name to use in grant table entries. The fourth argument is 1 if the script is being run to create tables for use on Windows, 0 otherwise.

4.4.4 `mysqlbug` — Generate Bug Report

This program enables you to generate a bug report and send it to Oracle Corporation. It is a shell script and runs on Unix.

The normal way to report bugs is to visit http://bugs.mysql.com/, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports. If you have no Web access, you can generate a bug report by using the `mysqlbug` script.

`mysqlbug` helps you generate a report by determining much of the following information automatically, but if something important is missing, please include it with your message. `mysqlbug` can be found in the `scripts` directory (source distribution) and in the `bin` directory under your MySQL installation directory (binary distribution).

Invoke `mysqlbug` without arguments:

```
shell> mysqlbug
```

The script will place you in an editor with a copy of the report to be sent. Edit the lines near the beginning that indicate the nature of the problem. Then write the file to save your changes, quit the editor, and `mysqlbug` will send the report by email.

4.4.5 `mysql_fix_privilege_tables` — Upgrade MySQL System Tables

Some releases of MySQL introduce changes to the structure of the system tables in the `mysql` database to add new privileges or support new features. When you update to a new version of MySQL, you should update your system tables as well to make sure that their structure is up to date. Otherwise, there might be capabilities that you cannot take advantage of. First, make a backup of your `mysql` database, and then use the following procedure.

On Unix or Unix-like systems, update the system tables by running the `mysql_fix_privilege_tables` script:

```
shell> mysql_fix_privilege_tables
```

You must run this script while the server is running. It attempts to connect to the server running on the local host as `root`. If your `root` account requires a password, indicate the password on the command line. For MySQL 4.1 and up, specify the password like this:

```
shell> mysql_fix_privilege_tables --password=root_password
```

Prior to MySQL 4.1, specify the password like this:

```
shell> mysql_fix_privilege_tables root_password
```

The `mysql_fix_privilege_tables` script performs any actions necessary to convert your system tables to the current format. You might see some `Duplicate column name` warnings as it runs; you can ignore them.

After running the script, stop the server and restart it so that it uses any changes that were made to the system tables.

On Windows systems, there isn't an easy way to update the system tables until MySQL 4.0.15. From version 4.0.15 on, MySQL distributions include a `mysql_fix_privilege_tables.sql` SQL script that

you can run using the mysql client. For example, if your MySQL installation is located at C:\Program Files\MySQL\MySQL Server 4.1, the commands look like this:

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 4.1"
C:\> bin\mysql -u root -p mysql
mysql> SOURCE scripts/mysql_fix_privilege_tables.sql
```

The mysql command will prompt you for the root password; enter it when prompted.

If your installation is located in some other directory, adjust the path names appropriately.

As with the Unix procedure, you might see some Duplicate column name warnings as mysql processes the statements in the mysql_fix_privilege_tables.sql script; you can ignore them.

After running the script, stop the server and restart it.

4.4.6 mysql_install_db — Initialize MySQL Data Directory

mysql_install_db initializes the MySQL data directory and creates the system tables that it contains, if they do not exist.

To invoke mysql_install_db, use the following syntax:

```
shell> mysql_install_db [options]
```

Because the MySQL server, mysqld, needs to access the data directory when it runs later, you should either run mysql_install_db from the same account that will be used for running mysqld or run it as root and use the --user [255] option to indicate the user name that mysqld will run as. It might be necessary to specify other options such as --basedir [254] or --datadir [254] if mysql_install_db does not use the correct locations for the installation directory or data directory. For example:

```
shell> bin/mysql_install_db --user=mysql \
         --basedir=/opt/mysql/mysql \
         --datadir=/opt/mysql/mysql/data
```

mysql_install_db supports the following options, which can be specified on the command line or in the [mysql_install_db] and (if they are common to mysqld) [mysqld] option file groups.

- --basedir=*path* [254]

  The path to the MySQL installation directory.

- --force [254]

  Cause mysql_install_db to run even if DNS does not work. In that case, grant table entries that normally use host names will use IP addresses.

- --datadir=*path* [254], --ldata=*path* [254]

  The path to the MySQL data directory.

- --rpm [254]

  For internal use. This option is used by RPM files during the MySQL installation process.

- `--skip-name-resolve` [255]

  Use IP addresses rather than host names when creating grant table entries. This option can be useful if your DNS does not work.

- `--user=user_name` [255]

  The login user name to use for running `mysqld`. Files and directories created by `mysqld` will be owned by this user. You must be `root` to use this option. By default, `mysqld` runs using your current login name and files and directories that it creates will be owned by you.

- `--verbose` [255]

  Verbose mode. Print more information about what the program does.

- `--windows` [255]

  For internal use. This option is used for creating Windows distributions.

4.4.7 `mysql_secure_installation` — Improve MySQL Installation Security

This program enables you to improve the security of your MySQL installation in the following ways:

- You can set a password for `root` accounts.
- You can remove `root` accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the `test` database (which by default can be accessed by all users, even anonymous users), and privileges that permit anyone to access databases with names that start with `test_`.

`mysql_secure_installation` helps you implement security recommendations similar to those described at Section 2.10.3, “Securing the Initial MySQL Accounts”.

Invoke `mysql_secure_installation` without arguments:

```
shell> mysql_secure_installation
```

The script will prompt you to determine which actions to perform.

`mysql_secure_installation` is not available on Windows.

4.4.8 `mysql_tzinfo_to_sql` — Load the Time Zone Tables

The `mysql_tzinfo_to_sql` program loads the time zone tables in the `mysql` database. It is used on systems that have a *zoneinfo* database (the set of files describing time zones). Examples of such systems are Linux, FreeBSD, Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a zoneinfo database, you can use the downloadable package described in Section 9.7, “MySQL Server Time Zone Support”.

`mysql_tzinfo_to_sql` can be invoked several ways:

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

For the first invocation syntax, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

The second syntax causes `mysql_tzinfo_to_sql` to load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, invoke `mysql_tzinfo_to_sql` using the third syntax, which initializes the leap second information. *`tz_file`* is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

After running `mysql_tzinfo_to_sql`, it is best to restart the server so that it does not continue to use any previously cached time zone data.

`mysql_tzinfo_to_sql` was added in MySQL 4.1.3.

4.5 MySQL Client Programs

This section describes client programs that connect to the MySQL server.

4.5.1 `mysql` — The MySQL Command-Line Tool

`mysql` is a simple SQL shell (with GNU `readline` capabilities). It supports interactive and noninteractive use. When used interactively, query results are presented in an ASCII-table format. When used noninteractively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the `--quick` [263] option. This forces `mysql` to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the `mysql_use_result()` C API function in the client/server library rather than `mysql_store_result()`.

Using `mysql` is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with “`;`”, `\g`, or `\G` and press Enter.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

On Unix, the mysql client logs statements executed interactively to a history file. See Section 4.5.1.3, "mysql Logging".

4.5.1.1 mysql Options

mysql supports the following options, which can be specified on the command line or in the [mysql] and [client] option file groups. mysql also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.2 mysql Options

Format	Option File	Description	Introduced	Deprecated
--auto-rehash [260]	auto-rehash [260]	Enable automatic rehashing		
--batch [260]	batch [260]	Don't use history file		
--character-sets-dir=path [260]	character-sets-dir [260]	Set the default character set		
--column-names [260]	column-names [260]	Write column names in results		
--compress [260]	compress [260]	Compress all information sent between the client and the server		
--connect_timeout=value [265]	connect_timeout [265]	The number of seconds before connection timeout		
--database=dbname [260]	database [260]	The database to use		
--debug[=debug_options] [260]	debug [260]	Write a debugging log		
--debug-info [260]	debug-info [260]	Print debugging information, memory and CPU statistics when the program exits		
--default-character-set=charset_name [260]	default-character-set [260]	Use charset_name as the default character set		
--delimiter=str [261]	delimiter [261]	Set the statement delimiter		
--execute=statement [261]	execute [261]	Execute the statement and quit		
--force [261]	force [261]	Continue even if an SQL error occurs		
--help [260]		Display help message and exit		
--host=host_name [261]	host [261]	Connect to the MySQL server on the given host		
--html [261]	html [261]	Produce HTML output		
--ignore-spaces [261]	ignore-spaces [261]	Ignore spaces after function names		
--line-numbers [261]	line-numbers [261]	Write line numbers for errors		
--local-infile[={0|1}] [261]	local-infile [261]	Enable or disable for LOCAL capability for LOAD DATA INFILE		

Format	Option File	Description	Introduced	Deprecated
--max_allowed_packet=value [265]	max_allowed_packet [265]	The maximum packet length to send to or receive from the server		
--max_join_size=value [265]	max_join_size [265]	The automatic limit for rows in a join when using --safe-updates		
--named-commands [261]	named-commands [261]	Enable named mysql commands		
--net_buffer_length=value [265]	net_buffer_length [265]	The buffer size for TCP/IP and socket communication		
--no-auto-rehash [261]		Disable automatic rehashing		4.1.0
--no-beep [261]	no-beep [261]	Do not beep when errors occur		
--no-named-commands [261]	no-named-commands [261]	Disable named mysql commands		4.1.0
--no-pager [262]	no-pager [262]	Deprecated form of --skip-pager		4.1.0
--no-tee [262]	no-tee [262]	Do not copy output to a file		
--one-database [262]	one-database [262]	Ignore statements except those for the default database named on the command line		
--pager[=command] [262]	pager [262]	Use the given command for paging query output		
--password[=password] [262]	password [262]	The password to use when connecting to the server		
--pipe [263]		On Windows, connect to server using a named pipe		
--port=port_num [263]	port [263]	The TCP/IP port number to use for the connection		
--prompt=format_str [263]	prompt [263]	Set the prompt to the specified format		
--protocol=type [263]	protocol [263]	The connection protocol to use		
--quick [263]	quick [263]	Do not cache each query result		
--raw [263]	raw [263]	Write column values without escape conversion		
--reconnect [264]	reconnect [264]	If the connection to the server is lost, automatically try to reconnect		
--safe-updates [264]	safe-updates [264]	Allow only UPDATE and DELETE statements that specify key values		
--secure-auth [264]	secure-auth [264]	Do not send passwords to the server in old (pre-4.1.1) format	4.1.1	
--select_limit=value [265]	select_limit [265]	The automatic limit for SELECT statements when using --safe-updates		
--sigint-ignore [264]	sigint-ignore [264]	Ignore SIGINT signals (typically the result of typing Control+C)	4.1.6	

Format	Option File	Description	Introduced	Deprecated
--silent [264]	silent [264]	Silent mode		
--skip-auto-rehash [260]	skip-auto-rehash [260]	Disable automatic rehashing		
--skip-column-names [264]	skip-column-names [264]	Do not write column names in results		
--skip-line-numbers [264]	skip-line-numbers [264]	Skip line numbers for errors		
--skip-named-commands [261]	skip-named-commands [261]	Disable named mysql commands		
--skip-pager [262]	skip-pager [262]	Disable paging		
--skip-reconnect [264]	skip-reconnect [264]	Disable reconnecting		
--socket=path [264]	socket [264]	For connections to localhost		
--ssl[=TRUE|FALSE] [264]		Enable an SSL connection to the server. This option is set to TRUE when any other SSL option is used, and so is normally not needed.		
--ssl-ca=file_name [264]	ssl-ca [264]	The path to a file that contains a list of trusted SSL CAs		
--ssl-capath=dir_name [264]	ssl-capath [264]	The path to a directory that contains trusted SSL CA certificates in PEM format		
--ssl-cert=file_name [264]	ssl-cert [264]	The name of the SSL certificate file to use for establishing a secure connection		
--ssl-cipher=cipher_list [264]	ssl-cipher [264]	A list of allowable ciphers to use for SSL encryption		
--ssl-key=file_name [264]	ssl-key [264]	The name of the SSL key file to use for establishing a secure connection		
--ssl-verify-server-cert [264]	ssl-verify-server-cert [264]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server		
--table [264]	table [264]	Display output in tabular format		
--tee=file_name [264]	tee [264]	Append a copy of output to the given file		
--unbuffered [265]	unbuffered [265]	Flush the buffer after each query		
--user=user_name [265]	user [265]	MySQL user name to use when connecting to server		
--verbose [265]		Verbose mode		
--version [265]		Display version information and exit		
--vertical [265]	vertical [265]	Print query output rows vertically (one line per column value)		

Format	Option File	Description	Introduced	Deprecated
--wait [265]	wait [265]	If the connection cannot be established, wait and retry instead of aborting		
--xml [265]	xml [265]	Produce XML output		

- `--help` [260], `-?`

  Display a help message and exit.

- `--auto-rehash` [260]

  Enable automatic rehashing. This option is on by default, which enables database, table, and column name completion. Use `--skip-auto-rehash` [260] to disable rehashing. That causes `mysql` to start faster, but you must issue the `rehash` command if you want to use name completion.

  To complete a name, enter the first part and press Tab. If the name is unambiguous, `mysql` completes it. Otherwise, you can press Tab again to see the possible names that begin with what you have typed so far. Completion does not occur if there is no default database.

- `--batch` [260], `-B`

  Print results using tab as the column separator, with each row on a new line. With this option, `mysql` does not use the history file.

  Batch mode results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` [263] option.

- `--character-sets-dir=`*`path`* [260]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--column-names` [260]

  Write column names in results.

- `--compress` [260], `-C`

  Compress all information sent between the client and the server if both support compression.

- `--database=`*`db_name`* [260], `-D` *`db_name`*

  The database to use. This is useful primarily in an option file.

- `--debug[=`*`debug_options`*`]` [260], `-#` `[`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:o,`*`file_name`*`'`. The default is `'d:t:o,/tmp/mysql.trace'`.

- `--debug-info` [260], `-T`

  Print some debugging information when the program exits.

- `--default-character-set=`*`charset_name`* [260]

  Use *`charset_name`* as the default character set for the client and connection.

  A common issue that can occur when the operating system uses `utf8` or another multi-byte character set is that output from the `mysql` client is formatted incorrectly, due to the fact that the MySQL client

uses the latin1 character set by default. You can usually fix such issues by using this option to force the client to use the system character set instead.

See Section 9.6, “Character Set Configuration”, for more information.

- --delimiter=*str* [261]

  Set the statement delimiter. The default is the semicolon character (“;”).

- --execute=*statement* [261], -e *statement*

  Execute the statement and quit. The default output format is like that produced with --batch [260]. See Section 4.2.3.1, “Using Options on the Command Line”, for some examples. With this option, mysql does not use the history file.

- --force [261], -f

  Continue even if an SQL error occurs.

- --host=*host_name* [261], -h *host_name*

  Connect to the MySQL server on the given host.

- --html [261], -H

  Produce HTML output.

- --ignore-spaces [261], -i

  Ignore spaces after function names. The effect of this is described in the discussion for the IGNORE_SPACE [458] SQL mode (see Section 5.1.6, “Server SQL Modes”).

- --line-numbers [261]

  Write line numbers for errors. Disable this with --skip-line-numbers [264].

- --local-infile[={0|1}] [261]

  Enable or disable LOCAL capability for LOAD DATA INFILE. With no value, the option enables LOCAL. The option may be given as --local-infile=0 [261] or --local-infile=1 [261] to explicitly disable or enable LOCAL. Enabling LOCAL has no effect if the server does not also support it.

- --named-commands [261], -G

  Enable named mysql commands. Long-format commands are permitted, not just short-format commands. For example, quit and \q both are recognized. Use --skip-named-commands [261] to disable named commands. See Section 4.5.1.2, “mysql Commands”.

- --no-auto-rehash [260], -A

  This has the same effect as -skip-auto-rehash. See the description for --auto-rehash [260].

- --no-beep [261], -b

  Do not beep when errors occur.

- --no-named-commands [261], -g

  Disable named commands. Use the * form only, or use named commands only at the beginning of a line ending with a semicolon (“;”). As of MySQL 3.23.22, mysql starts with this option *enabled*

by default. However, even with this option, long-format commands still work from the first line. See Section 4.5.1.2, “mysql Commands”.

- --no-pager [262]

  Deprecated form of --skip-pager [262]. See the --pager [262] option. --no-pager [262] is removed in MySQL 5.5.

- --no-tee [262]

  Deprecated form of --skip-tee [264]. See the --tee [264] option. --no-tee [262] is removed in MySQL 5.5.

- --one-database [262], -o

  Ignore statements except those that occur while the default database is the one named on the command line. This option is rudimentary and should be used with care. Statement filtering is based only on USE statements.

  Initially, mysql executes statements in the input because specifying a database *db_name* on the command line is equivalent to inserting USE *db_name* at the beginning of the input. Then, for each USE statement encountered, mysql accepts or rejects following statements depending on whether the database named is the one on the command line. The content of the statements is immaterial.

  Suppose that mysql is invoked to process this set of statements:

```
DELETE FROM db2.t2;
USE db2;
DROP TABLE db1.t1;
CREATE TABLE db1.t1 (i INT);
USE db1;
INSERT INTO t1 (i) VALUES(1);
CREATE TABLE db2.t1 (j INT);
```

  If the command line is mysql --force --one-database db1, mysql handles the input as follows:

  - The DELETE statement is executed because the default database is db1, even though the statement names a table in a different database.
  - The DROP TABLE and CREATE TABLE statements are not executed because the default database is not db1, even though the statements name a table in db1.
  - The INSERT and CREATE TABLE statements are executed because the default database is db1, even though the CREATE TABLE statement names a table in a different database.

- --pager[=*command*] [262]

  Use the given command for paging query output. If the command is omitted, the default pager is the value of your PAGER environment variable. Valid pagers are less, more, cat [> filename], and so forth. This option works only on Unix and only in interactive mode. To disable paging, use --skip-pager [262]. Section 4.5.1.2, “mysql Commands”, discusses output paging further.

- --password[=*password*] [262], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password [262] or -p option on the command line, mysql prompts for one.

Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe` [263], `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=port_num` [263], `-P port_num`

  The TCP/IP port number to use for the connection.

- `--prompt=format_str` [263]

  Set the prompt to the specified format. The default is `mysql>`. The special sequences that the prompt can contain are described in Section 4.5.1.2, “`mysql` Commands”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [263]

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”. This option was added in MySQL 4.1.

- `--quick` [263], `-q`

  Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, `mysql` does not use the history file.

- `--raw` [263], `-r`

  For tabular output, the “boxing” around columns enables one column value to be distinguished from another. For nontabular output (such as is produced in batch mode or when the `--batch` [260] or `--silent` [264] option is given), special characters are escaped in the output so they can be identified easily. Newline, tab, `NUL`, and backslash are written as `\n`, `\t`, `\0`, and `\\`. The `--raw` [263] option disables this character escaping.

  The following example demonstrates tabular versus nontabular output and the use of raw mode to disable escaping:

```
% mysql
mysql> SELECT CHAR(92);
+----------+
| CHAR(92) |
+----------+
| \        |
+----------+

% mysql -s
mysql> SELECT CHAR(92);
CHAR(92)
\\

% mysql -s -r
mysql> SELECT CHAR(92);
CHAR(92)
\
```

- --reconnect [264]

  If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use --skip-reconnect [264]. Added in MySQL 4.1.0.

- --safe-updates [264], --i-am-a-dummy [264], -U

  Permit only those UPDATE and DELETE statements that specify which rows to modify by using key values. If you have set this option in an option file, you can override it by using --safe-updates [264] on the command line. See Section 4.5.1.6, "mysql Tips", for more information about this option.

- --secure-auth [264]

  Do not send passwords to the server in old (pre-4.1.1) format. This prevents connections except for servers that use the newer password format. This option was added in MySQL 4.1.1.

- --sigint-ignore [264]

  Ignore SIGINT signals (typically the result of typing Control-C). This option was added in MySQL 4.1.6.

- --silent [264], -s

  Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

  This option results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the --raw [263] option.

- --skip-column-names [264], -N

  Do not write column names in results.

- --skip-line-numbers [264], -L

  Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- --socket=*path* [264], -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

  Options that begin with --ssl [521] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, "SSL Command Options".

- --table [264], -t

  Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- --tee=*file_name* [264]

  Append a copy of output to the given file. This option works only in interactive mode. Section 4.5.1.2, "mysql Commands", discusses tee files further.

- `--unbuffered` [265], `-n`

  Flush the buffer after each query.

- `--user=user_name` [265], `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose` [265], `-v`

  Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces table output format even in batch mode.)

- `--version` [265], `-V`

  Display version information and exit.

- `--vertical` [265], `-E`

  Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with `\G`.

- `--wait` [265], `-w`

  If the connection cannot be established, wait and retry instead of aborting.

- `--xml` [265], `-X`

  Produce XML output.

You can also set the following variables by using `--var_name=value` syntax:

- `connect_timeout`

  The number of seconds before connection timeout. (Default value is `0`.)

- `max_allowed_packet`

  The maximum size of the buffer for client/server communication. The default is 16MB, the maximum is 1GB.

- `max_join_size`

  The automatic limit for rows in a join when using `--safe-updates` [264]. (Default value is 1,000,000.)

- `net_buffer_length`

  The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

  The automatic limit for `SELECT` statements when using `--safe-updates` [264]. (Default value is 1,000.)

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. In MySQL 4.1, this syntax is deprecated.

4.5.1.2 mysql Commands

mysql sends each SQL statement that you issue to the server to be executed. There is also a set of commands that mysql itself interprets. For a list of these commands, type help or \h at the mysql> prompt:

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear command.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set query delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given
               outfile.
use       (\u) Use another database. Takes database name as argument.
charset_name(\C) Switch to another charset. Might be needed for processing
               binlog.

For server side help, type 'help contents'
```

Each command has both a long and short form. The long form is not case sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

The use of short-form commands within multi-line /* ... */ comments is not supported.

- help [*arg*], \h [*arg*], \? [*arg*], ? [*arg*]

  Display a help message listing the available mysql commands.

  If you provide an argument to the help command, mysql uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see Section 4.5.1.4, “mysql Server-Side Help”.

- charset_name *charset_name*, \C *charset_name*

  Change the default character set and issue a SET NAMES statement. This enables the character set to remain synchronized on the client and server if mysql is run with auto-reconnect enabled (which is not recommended), because the specified character set is used for reconnects.

  This command was added in MySQL 4.1.19. In MySQL 5.0 and up, the command name is charset

- clear, \c

  Clear the current input. Use this if you change your mind about executing the statement that you are entering.

- `connect [db_name host_name]`, `\r [db_name host_name]`

  Reconnect to the server. The optional database name and host name arguments may be given to specify the default database or the host where the server is running. If omitted, the current values are used.

- `delimiter str`, `\d str`

  Change the string that `mysql` interprets as the separator between SQL statements. The default is the semicolon character (“`;`”).

  The delimiter can be specified as an unquoted or quoted argument. Quoting can be done with either single quote (`'`) or douple quote (`"`) characters. To include a quote within a quoted string, either quote the string with the other quote character or escape the quote with a backslash (“`\`”) character. Backslash should be avoided outside of quoted strings because it is the escape character for MySQL. For an unquoted argument, the delmiter is read up to the first space or end of line. For a quoted argument, the delimiter is read up to the matching quote on the line.

  When the delimiter recognized by `mysql` is set to something other than the default of “`;`”, instances of that character are sent to the server without interpretation. However, the server itself still interprets “`;`” as a statement delimiter and processes statements accordingly. This behavior on the server side comes into play for multiple-statement execution (see Section 17.6.15, “C API Support for Multiple Statement Execution”).

- `edit`, `\e`

  Edit the current input statement. `mysql` checks the values of the `EDITOR` and `VISUAL` environment variables to determine which editor to use. The default editor is `vi` if neither variable is set.

  The `edit` command works only in Unix.

- `ego`, `\G`

  Send the current statement to the server to be executed and display the result using vertical format.

- `exit`, `\q`

  Exit `mysql`.

- `go`, `\g`

  Send the current statement to the server to be executed.

- `nopager`, `\n`

  Disable output paging. See the description for `pager`.

  The `nopager` command works only in Unix.

- `notee`, `\t`

  Disable output copying to the tee file. See the description for `tee`.

- `pager [command]`, `\P [command]`

  Enable output paging. By using the `--pager` [262] option when you invoke `mysql`, it is possible to browse or search query results in interactive mode with Unix programs such as `less`, `more`, or any

other similar program. If you specify no value for the option, mysql checks the value of the PAGER environment variable and sets the pager to that. Pager functionality works only in interactive mode.

Output paging can be enabled interactively with the pager command and disabled with nopager. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or stdout if no pager was specified.

Output paging works only in Unix because it uses the popen() function, which does not exist on Windows. For Windows, the tee option can be used instead to save query output, although it is not as convenient as pager for browsing output in some situations.

- print, \p

  Print the current input statement without executing it.

- prompt [*str*], \R [*str*]

  Reconfigure the mysql prompt to the given string. The special character sequences that can be used in the prompt are described later in this section.

  If you specify the prompt command with no argument, mysql resets the prompt to the default of mysql>.

- quit, \q

  Exit mysql.

- rehash, \#

  Rebuild the completion hash that enables database, table, and column name completion while you are entering statements. (See the description for the --auto-rehash [260] option.)

- source *file_name*, \. *file_name*

  Read the named file and executes the statements contained therein. On Windows, you can specify path name separators as / or \\.

- status, \s

  Provide status information about the connection and the server you are using. If you are running in --safe-updates [264] mode, status also prints the values for the mysql variables that affect your queries.

- system *command*, \! *command*

  Execute the given command using your default command interpreter.

  The system command works only in Unix.

- tee [*file_name*], \T [*file_name*]

  By using the --tee [264] option when you invoke mysql, you can log statements and their output. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. mysql flushes results to the file after each statement, just before it prints its next prompt. Tee functionality works only in interactive mode.

  You can enable this feature interactively with the tee command. Without a parameter, the previous file is used. The tee file can be disabled with the notee command. Executing tee again re-enables logging.

- `use db_name`, `\u db_name`

  Use *db_name* as the default database.

Here are a few tips about the `pager` command:

- You can use it to write to a file and the results go only to the file:

```
mysql> pager cat > /tmp/log.txt
```

  You can also pass any options for the program that you want to use as your pager:

```
mysql> pager less -n -i -S
```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to `less` can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within `less` to switch the horizontal-browse mode on and off. For more information, read the `less` manual page:

```
shell> man less
```

- The `-F` and `-X` options may be used with `less` to cause it to exit if output fits on one screen, which is convenient when no scrolling is necessary:

```
mysql> pager less -n -i -S -F -X
```

- You can specify very complex pager commands for handling query output:

```
mysql> pager cat | tee /dr1/tmp/res.txt \
         | tee /dr2/tmp/res2.txt | less -n -i -S
```

  In this example, the command would send query results to two files in two different directories on two different file systems mounted on `/dr1` and `/dr2`, yet still display the results onscreen using `less`.

You can also combine the `tee` and `pager` functions. Have a `tee` file enabled and `pager` set to `less`, and you are able to browse the results using the `less` program and still have everything appended into a file the same time. The difference between the Unix `tee` used with the `pager` command and the `mysql` built-in `tee` command is that the built-in `tee` works even if you do not have the Unix `tee` available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix `tee` used with `pager` does not log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within `mysql`. This is useful when you want to log some queries to a file, but not others.

From MySQL 4.0.2 on, the `prompt` command reconfigures the default `mysql>` prompt. The string for defining the prompt can contain the following special sequences.

Option	Description
`\c`	A counter that increments for each statement you issue
`\D`	The full current date
`\d`	The default database
`\h`	The server host
`\m`	Minutes of the current time

Option	Description
\n	A newline character
\O	The current month in three-letter format (Jan, Feb, …)
\o	The current month in numeric format
\P	am/pm
\p	The current TCP/IP port or socket file
\R	The current time, in 24-hour military time (0–23)
\r	The current time, standard 12-hour time (1–12)
\S	Semicolon
\s	Seconds of the current time
\t	A tab character
\U	Your full *user_name@host_name* account name
\u	Your user name
\v	The server version
\w	The current day of the week in three-letter format (Mon, Tue, …)
\Y	The current year, four digits
\y	The current year, two digits
_	A space
\	A space (a space follows the backslash)
\'	Single quote
\"	Double quote
\\	A literal "\" backslash character
\x	*x*, for any "*x*" not listed above

You can set the prompt in several ways:

- *Use an environment variable.* You can set the MYSQL_PS1 environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\u@\h) [\d]> "
```

- *Use a command-line option.* You can set the --prompt [263] option on the command line to mysql. For example:

```
shell> mysql --prompt="(\u@\h) [\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the prompt option in the [mysql] group of any MySQL option file, such as /etc/my.cnf or the .my.cnf file in your home directory. For example:

```
[mysql]
prompt=(\\u@\\h) [\\d]>\\_
```

  In this example, note that the backslashes are doubled. If you set the prompt using the prompt option in an option file, it is advisable to double the backslashes when using the special prompt options. There is

some overlap in the set of permissible prompt options and the set of special escape sequences that are recognized in option files. (The rules for escape sequences in option files are listed in Section 4.2.3.3, “Using Option Files”.) The overlap may cause you problems if you use single backslashes. For example, `\s` is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in `HH:MM:SS>` format:

```
[mysql]
prompt="\\r:\\m:\\s> "
```

- *Set the prompt interactively.* You can change your prompt interactively by using the `prompt` (or `\R`) command. For example:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

4.5.1.3 mysql Logging

On Unix, the `mysql` client logs statements executed interactively to a history file. By default, this file is named `.mysql_history` in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.

How Logging Occurs

Statement logging occurs as follows:

- Statements are logged only when executed interactively. Statements are noninteractive, for example, when read from a file or a pipe. It is also possible to suppress statement logging by using the `--batch` [260] or `--execute` [261] option.
- `mysql` logs each nonempty statement line individually.
- If a statement spans multiple lines (not including the terminating delimiter), `mysql` concatenates the lines to form the complete statement, maps newlines to spaces, and logs the result, plus a delimiter.

Consequently, an input statement that spans multiple lines can be logged twice. Consider this input:

```
mysql> SELECT
    -> 'Today is'
    -> ,
    -> CONCAT()
    -> ;
```

In this case, `mysql` logs the “SELECT”, “'Today is'”, “,”, “CONCAT()”, and “;” lines as it reads them. It also logs the complete statement, after mapping `SELECT\n'Today is'\n,\nCURDATE()` to `SELECT 'Today is' , CURDATE()`, plus a delimiter. Thus, these lines appear in logged output:

```
SELECT
'Today is'
,
CURDATE()
;
SELECT 'Today is' , CURDATE();
```

Controlling the History File

The `.mysql_history` file should be protected with a restrictive access mode because sensitive information might be written to it, such as the text of SQL statements that contain passwords. See Section 5.4.2.2, “End-User Guidelines for Password Security”.

If you do not want to maintain a history file, first remove `.mysql_history` if it exists. Then use either of the following techniques to prevent it from being created again:

- Set the `MYSQL_HISTFILE` environment variable to `/dev/null`. To cause this setting to take effect each time you log in, put it in one of your shell's startup files.
- Create `.mysql_history` as a symbolic link to `/dev/null`; this need be done only once:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

4.5.1.4 `mysql` Server-Side Help

```
mysql> help search_string
```

As of MySQL 4.1, if you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information (see Section 5.1.7, “Server-Side Help”).

If there is no match for the search string, the search fails:

```
mysql> help me

Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

Use `help contents` to see a list of the help categories:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
   Account Management
   Administration
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Language Structure
   Storage Engines
   Table Maintenance
   Transactions
```

If the search string matches multiple items, `mysql` shows a list of matching topics:

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
```

```
where <item> is one of the following topics:
   SHOW
   SHOW BINARY LOGS
   SHOW ENGINE
   SHOW LOGS
```

Use a topic as the search string to see the help entry for that topic:

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [purge-binary-logs], that shows how
to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

4.5.1.5 Executing SQL Statements from a Text File

The `mysql` client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell `mysql` to read its input from that file. To do so, create a text file *`text_file`* that contains the statements you wish to execute. Then invoke `mysql` as shown here:

```
shell> mysql db_name < text_file
```

If you place a `USE` *`db_name`* statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running `mysql`, you can execute an SQL script file using the `source` command or `\.` command:

```
mysql> source file_name
mysql> \. file_name
```

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs `<info_to_display>`.

You can also invoke mysql with the --verbose [265] option, which causes each statement to be displayed before the result that it produces.

For more information about batch mode, see Section 3.5, “Using mysql in Batch Mode”.

4.5.1.6 mysql Tips

This section describes some techniques that can help you use mysql more effectively.

Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with \G instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
*************************** 1. row ***************************
  msg_nro: 3068
     date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
    reply: monty@no.spam.com
  mail_to: "Thimble Smith" <tim@no.spam.com>
      sbj: UTF-8
      txt: >>>>> "Thimble" == Thimble Smith writes:

Thimble> Hi.  I think this is a good idea.  Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
     file: inbox-jani-1
     hash: 190402944
1 row in set (0.09 sec)
```

Using the --safe-updates [264] Option

For beginners, a useful startup option is --safe-updates [264] (or --i-am-a-dummy [264], which has the same effect). This option was introduced in MySQL 3.23.11. It is helpful for cases when you might have issued a DELETE FROM *tbl_name* statement but forgotten the WHERE clause. Normally, such a statement deletes all rows from the table. With --safe-updates [264], you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the --safe-updates [264] option, mysql issues the following statement when it connects to the MySQL server:

```
SET sql_safe_updates=1, sql_select_limit=1000, sql_max_join_size=1000000;
```

See Section 5.1.3, “Server System Variables”.

The SET statement has the following effects:

- You are not permitted to execute an UPDATE or DELETE statement unless you specify a key constraint in the WHERE clause or provide a LIMIT clause (or both). For example:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;

UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- The server limits all large SELECT results to 1,000 rows unless the statement includes a LIMIT clause.
- The server aborts multiple-table SELECT statements that probably need to examine more than 1,000,000 row combinations.

To specify limits different from 1,000 and 1,000,000, you can override the defaults by using the --select_limit and --max_join_size options:

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

Disabling mysql Auto-Reconnect

If the mysql client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if mysql succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+------+
| a    |
+------+
| NULL |
+------+
1 row in set (0.05 sec)
```

The @a user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have mysql terminate with an error if the connection has been lost, you can start the mysql client with the --skip-reconnect [264] option.

For more information about auto-reconnect and its effect on state information when a reconnection occurs, see Section 17.6.14, “Controlling Automatic Reconnection Behavior”.

4.5.2 mysqladmin — Client for Administering a MySQL Server

mysqladmin is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke mysqladmin like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

mysqladmin supports the following commands. Some of the commands take an argument following the command name.

- create *db_name*

  Create a new database named *db_name*.

- debug

  Tell the server to write debug information to the error log. Format and content of this information is subject to change.

- drop *db_name*

  Delete the database named *db_name* and all its tables.

- extended-status

  Display the server status variables and their values.

- flush-hosts

  Flush all information in the host cache.

- flush-logs

  Flush all logs.

- flush-privileges

  Reload the grant tables (same as reload).

- flush-status

  Clear status variables.

- flush-tables

  Flush all tables.

- flush-threads

  Flush the thread cache. (Added in MySQL 3.23.16.)

- kill *id,id,...*

  Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

- old-password *new-password*

  This is like the password command but stores the password using the old (pre-4.1) password-hashing format. This command was added in MySQL 4.1.0. (See Section 5.4.2.3, “Password Hashing in MySQL”.)

- password *new-password*

  Set a new password. This changes the password to *new-password* for the account that you use with mysqladmin for connecting to the server. Thus, the next time you invoke mysqladmin (or any other client program) using the same account, you will need to specify the new password.

If the *new-password* value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotation marks. On Windows, be sure to use double quotation marks rather than single quotation marks; single quotation marks are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

Caution

Do not use this command used if the server was started with the `--skip-grant-tables` [392] option. No password change will be applied. This is true even if you precede the `password` command with `flush-privileges` on the same command line to re-enable the grant tables because the flush operation occurs after you connect. However, you can use `mysqladmin flush-privileges` to re-enable the grant table and then use a separate `mysqladmin password` command to change the password.

- `ping`

  Check whether the server is available. The return status from `mysqladmin` is 0 if the server is running, 1 if it is not. Beginning with MySQL 4.0.22, the status is 0 even in case of an error such as `Access denied`, because that means the server is running but refused the connection, which is different from the server not running.

- `processlist`

  Show a list of active server threads. This is like the output of the `SHOW PROCESSLIST` statement. If the `--verbose` [281] option is given, the output is like that of `SHOW FULL PROCESSLIST`. (See Section 12.4.5.19, “`SHOW PROCESSLIST` Syntax”.)

- `reload`

  Reload the grant tables.

- `refresh`

  Flush all tables and close and open log files.

- `shutdown`

  Stop the server.

- `start-slave`

  Start replication on a slave server. (Added in MySQL 3.23.16.)

- `status`

  Display a short server status message.

- `stop-slave`

  Stop replication on a slave server. (Added in MySQL 3.23.16.)

- `variables`

  Display the server system variables and their values.

- version

  Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+---------+------+-------+------------------+
| Id | User  | Host      | db | Command | Time | State | Info             |
+----+-------+-----------+----+---------+------+-------+------------------+
| 51 | monty | localhost |    | Query   | 0    |       | show processlist |
+----+-------+-----------+----+---------+------+-------+------------------+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

The mysqladmin status command result displays the following values:

- Uptime [457]

  The number of seconds the MySQL server has been running.

- Threads

  The number of active threads (clients).

- Questions [454]

  The number of questions (queries) from clients since the server was started.

- Slow queries

  The number of queries that have taken more than long_query_time [417] seconds. See Section 5.3.5, “The Slow Query Log”.

- Opens

  The number of tables the server has opened.

- Flush tables

  The number of flush-*, refresh, and reload commands the server has executed.

- Open tables

  The number of tables that currently are open.

- Memory in use

  The amount of memory allocated directly by mysqld. This value is displayed only when MySQL has been compiled with --with-debug=full [98].

- Maximum memory used

  The maximum amount of memory allocated directly by mysqld. This value is displayed only when MySQL has been compiled with --with-debug=full [98].

If you execute mysqladmin shutdown when connecting to a local server using a Unix socket file, mysqladmin waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

mysqladmin supports the following options, which can be specified on the command line or in the [mysqladmin] and [client] option file groups. mysqladmin also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.3 mysqladmin Options

Format	Option File	Description	Introduced
--compress [280]	compress [280]	Compress all information sent between the client and the server	
--connect_timeout=seconds [282]	connect_timeout [282]	The number of seconds before connection timeout	
--count=# [280]	count [280]	The number of iterations to make for repeated command execution	
--debug[=debug_options] [280]	debug [280]	Write a debugging log	
--default-character-set=charset_name [280]	default-character-set [280]	Use charset_name as the default character set	4.1.9
--force [280]	force [280]	Continue even if an SQL error occurs	
--help [280]		Display help message and exit	
--host=host_name [280]	host [280]	Connect to the MySQL server on the given host	
--password[=password] [280]	password [280]	The password to use when connecting to the server	
--pipe [281]		On Windows, connect to server using a named pipe	
--port=port_num [281]	port [281]	The TCP/IP port number to use for the connection	
--protocol=type [281]	protocol [281]	The connection protocol to use	
--relative [281]	relative [281]	Show the difference between the current and previous values when used with the --sleep option	
--shutdown_timeout=seconds [282]	shutdown_timeout [282]	The maximum number of seconds to wait for server shutdown	
--silent [281]	silent [281]	Silent mode	
--sleep=delay [281]	sleep [281]	Execute commands repeatedly, sleeping for delay seconds in between	
--socket=path [281]	socket [281]	For connections to localhost	
--ssl-ca=file_name [281]	ssl-ca [281]	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name [281]	ssl-capath [281]	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name [281]	ssl-cert [281]	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list [281]	ssl-cipher [281]	A list of allowable ciphers to use for SSL encryption	

Format	Option File	Description	Introduced
--ssl-key=file_name [281]	ssl-key [281]	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert [281]	ssl-verify-server-cert [281]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--user=user_name, [281]	user [281]	MySQL user name to use when connecting to server	
--verbose [281]		Verbose mode	
--version [281]		Display version information and exit	
--vertical [282]	vertical [282]	Print query output rows vertically (one line per column value)	
--wait [282]	wait [282]	If the connection cannot be established, wait and retry instead of aborting	

- `--help` [280], `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`path`* [280]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--compress` [280], `-C`

  Compress all information sent between the client and the server if both support compression.

- `--count=`*`N`* [280], `-c` *`N`*

  The number of iterations to make for repeated command execution if the `--sleep` [281] option is given.

- `--debug[=`*`debug_options`*`]` [280], `-#` `[`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:o,`*`file_name`*`'`. The default is `'d:t:o,/tmp/mysqladmin.trace'`.

- `--default-character-set=`*`charset_name`* [280]

  Use *`charset_name`* as the default character set. See Section 9.6, “Character Set Configuration”. Added in MySQL 4.1.9.

- `--force` [280], `-f`

  Do not ask for confirmation for the `drop` *`db_name`* command. With multiple commands, continue even if an error occurs.

- `--host=`*`host_name`* [280], `-h` *`host_name`*

  Connect to the MySQL server on the given host.

- `--password[=`*`password`*`]` [280], `-p[`*`password`*`]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` [280] or `-p` option on the command line, `mysqladmin` prompts for one.

Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe` [281], `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=`*`port_num`* [281], `-P` *`port_num`*

  The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [281]

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”. This option was added in MySQL 4.1.

- `--relative` [281], `-r`

  Show the difference between the current and previous values when used with the `--sleep` [281] option. This option works only with the `extended-status` command.

- `--silent` [281], `-s`

  Exit silently if a connection to the server cannot be established.

- `--sleep=`*`delay`* [281], `-i` *`delay`*

  Execute commands repeatedly, sleeping for *`delay`* seconds in between. The `--count` [280] option determines the number of iterations. If `--count` [280] is not given, `mysqladmin` executes commands indefinitely until interrupted.

- `--socket=`*`path`* [281], `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` [521] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, “SSL Command Options”.

- `--user=`*`user_name`* [281], `-u` *`user_name`*

  The MySQL user name to use when connecting to the server.

- `--verbose` [281], `-v`

  Verbose mode. Print more information about what the program does.

- `--version` [281], `-V`

Display version information and exit.

- `--vertical` [282], `-E`

  Print output vertically. This is similar to `--relative` [281], but prints output vertically.

- `--wait[=count]` [282], `-w[count]`

  If the connection cannot be established, wait and retry instead of aborting. If a *`count`* value is given, it indicates the number of times to retry. The default is one time.

You can also set the following variables by using `--var_name=value` syntax:

- `connect_timeout`

  The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `shutdown_timeout`

  The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

4.5.3 mysqlcheck — A Table Maintenance Program

The `mysqlcheck` client performs table maintenance: It checks, repairs, optimizes, and analyzes tables.

Each table is locked and therefore unavailable to other sessions while it is being processed, although for check operations, the table is locked with a `READ` lock only (see Section 12.3.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax”, for more information about `READ` and `WRITE` locks). Table maintenance operations can be time-consuming, particularly for large tables. If you use the `--databases` or `--all-databases` option to process all tables in one or more databases, an invocation of `mysqlcheck` might take a long time. `mysqlcheck` is available as of MySQL 3.23.38.

`mysqlcheck` is similar in function to `myisamchk`, but works differently. The main operational difference is that `mysqlcheck` must be used when the `mysqld` server is running, whereas `myisamchk` should be used when it is not. The benefit of using `mysqlcheck` is that you do not have to stop the server to check or repair your tables.

`mysqlcheck` uses the SQL statements `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, and `OPTIMIZE TABLE` in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in Section 12.4.2, “Table Maintenance Statements”.

The `MyISAM` storage engine supports all four maintenance operations, so `mysqlcheck` can be used to perform any of them on `MyISAM` tables. Other storage engines do not necessarily support all operations. In such cases, an error message is displayed. For example, if `test.t` is a `MEMORY` table, an attempt to check it produces this result:

```
shell> mysqlcheck test t
test.t
note     : The storage engine for the table doesn't support check
```

If mysqlcheck is unable to repair a table, see Section 2.11.4, “Rebuilding or Repairing Tables or Indexes” for manual table repair strategies. This will be the case, for example, for InnoDB tables, which can be checked with CHECK TABLE, but not repaired with REPAIR TABLE.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

There are three general ways to invoke mysqlcheck:

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following *db_name* or if you use the --databases [285] or --all-databases [285] option, entire databases are checked.

mysqlcheck has a special feature compared to other client programs. The default behavior of checking tables (--check [285]) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of mysqlcheck named mysqlrepair, or make a symbolic link to mysqlcheck named mysqlrepair. If you invoke mysqlrepair, it repairs tables.

The names shown in the following table can be used to change mysqlcheck default behavior.

Command	Meaning
mysqlrepair	The default option is --repair [287]
mysqlanalyze	The default option is --analyze [285]
mysqloptimize	The default option is --optimize [286]

mysqlcheck supports the following options, which can be specified on the command line or in the [mysqlcheck] and [client] option file groups. mysqlcheck also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.4 mysqlcheck Options

Format	Option File	Description
--all-databases [285]	all-databases [285]	Check all tables in all databases
--all-in-1 [285]	all-in-1 [285]	Execute a single statement for each database that names all the tables from that database
--analyze [285]	analyze [285]	Analyze the tables
--auto-repair [285]	auto-repair [285]	If a checked table is corrupted, automatically fix it
--character-sets-dir=path [285]	character-sets-dir [285]	The directory where character sets are installed
--check [285]	check [285]	Check the tables for errors
--check-only-changed [285]	check-only-changed [285]	Check only tables that have changed since the last check
--compress [285]	compress [285]	Compress all information sent between the client and the server
--databases [285]	databases [285]	Process all tables in the named databases

Format	Option File	Description
--debug[=debug_options] [285]	debug [285]	Write a debugging log
--default-character-set=charset_name [285]	default-character-set [285]	Use charset_name as the default character set
--extended [286]	extended [286]	Check and repair tables
--fast [286]	fast [286]	Check only tables that have not been closed properly
--force [286]	force [286]	Continue even if an SQL error occurs
--help [285]		Display help message and exit
--host=host_name [286]	host [286]	Connect to the MySQL server on the given host
--medium-check [286]	medium-check [286]	Do a check that is faster than an --extended operation
--optimize [286]	optimize [286]	Optimize the tables
--password[=password] [286]	password [286]	The password to use when connecting to the server
--pipe [286]		On Windows, connect to server using a named pipe
--port=port_num [286]	port [286]	The TCP/IP port number to use for the connection
--protocol=type [286]	protocol [286]	The connection protocol to use
--quick [286]	quick [286]	The fastest method of checking
--repair [287]	repair [287]	Perform a repair that can fix almost anything except unique keys that are not unique
--silent [287]	silent [287]	Silent mode
--socket=path [287]	socket [287]	For connections to localhost
--ssl-ca=file_name [287]	ssl-ca [287]	The path to a file that contains a list of trusted SSL CAs
--ssl-capath=dir_name [287]	ssl-capath [287]	The path to a directory that contains trusted SSL CA certificates in PEM format
--ssl-cert=file_name [287]	ssl-cert [287]	The name of the SSL certificate file to use for establishing a secure connection
--ssl-cipher=cipher_list [287]	ssl-cipher [287]	A list of allowable ciphers to use for SSL encryption
--ssl-key=file_name [287]	ssl-key [287]	The name of the SSL key file to use for establishing a secure connection
--ssl-verify-server-cert [287]	ssl-verify-server-cert [287]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server
--tables [287]	tables [287]	Overrides the --databases or -B option
--use-frm [287]	use-frm [287]	For repair operations on MyISAM tables
--user=user_name, [287]	user [287]	MySQL user name to use when connecting to server

Format	Option File	Description
--verbose [287]		Verbose mode
--version [287]		Display version information and exit

- `--help` [285], `-?`

  Display a help message and exit.

- `--all-databases` [285], `-A`

  Check all tables in all databases. This is the same as using the `--databases` [285] option and naming all the databases on the command line.

- `--all-in-1` [285], `-1`

  Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze` [285], `-a`

  Analyze the tables.

- `--auto-repair` [285]

  If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--character-sets-dir=`*`path`* [285]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--check` [285], `-c`

  Check the tables for errors. This is the default operation.

- `--check-only-changed` [285], `-C`

  Check only tables that have changed since the last check or that have not been closed properly.

- `--compress` [285]

  Compress all information sent between the client and the server if both support compression.

- `--databases` [285], `-B`

  Process all tables in the named databases. Normally, `mysqlcheck` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names.

- `--debug[=`*`debug_options`*`]` [285], `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:o,`*`file_name`*`'`. The default is `'d:t:o'`.

- `--default-character-set=`*`charset_name`* [285]

  Use *`charset_name`* as the default character set. See Section 9.6, “Character Set Configuration”.

- `--extended` [286], `-e`

  If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

  If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--fast` [286], `-F`

  Check only tables that have not been closed properly.

- `--force` [286], `-f`

  Continue even if an SQL error occurs.

- `--host=host_name` [286], `-h host_name`

  Connect to the MySQL server on the given host.

- `--medium-check` [286], `-m`

  Do a check that is faster than an `--extended` [286] operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--optimize` [286], `-o`

  Optimize the tables.

- `--password[=password]` [286], `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` [286] or `-p` option on the command line, `mysqlcheck` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe` [286], `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=port_num` [286], `-P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [286]

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”. This option was added in MySQL 4.1.

- `--quick` [286], `-q`

  If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- `--repair` [287], `-r`

  Perform a repair that can fix almost anything except unique keys that are not unique.

- `--silent` [287], `-s`

  Silent mode. Print only error messages.

- `--socket=path` [287], `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` [521] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, “SSL Command Options”.

- `--tables` [287]

  Override the `--databases` [285] or `-B` option. All name arguments following the option are regarded as table names.

- `--use-frm` [287]

  For repair operations on `MyISAM` tables, get the table structure from the `.frm` file so that the table can be repaired even if the `.MYI` header is corrupted. This option was added in MySQL 4.0.5.

- `--user=user_name` [287], `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose` [287], `-v`

  Verbose mode. Print information about the various stages of program operation.

- `--version` [287], `-V`

  Display version information and exit.

4.5.4 `mysqldump` — A Database Backup Program

The `mysqldump` client is a backup program originally written by Igor Romanenko. It can be used to dump a database or a collection of databases for backup or transfer to another SQL server (not necessarily a MySQL server). The dump typically contains SQL statements to create the table, populate it, or both. However, `mysqldump` can also be used to generate files in CSV, other delimited text, or XML format.

If you are doing a backup on the server and your tables all are `MyISAM` tables, consider using the `mysqlhotcopy` instead because it can accomplish faster backups and faster restores. See Section 4.6.8, “`mysqlhotcopy` — A Database Backup Program”.

There are three general ways to invoke `mysqldump`:

```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
```

```
shell> mysqldump [options] --all-databases
```

If you do not name any tables following *db_name* or if you use the --databases [294] or --all-databases [293] option, entire databases are dumped.

To see a list of the options your version of mysqldump supports, execute mysqldump --help.

Some mysqldump options are shorthand for groups of other options:

- Use of --opt [297] is the same as specifying --add-drop-table [293], --add-locks [293], --create-options [294], --disable-keys [294], --extended-insert [294], --lock-tables [295], --quick [298], and --set-charset [298]. As of MySQL 4.1, all of the options that --opt [297] stands for also are on by default because --opt [297] is on by default.
- Use of --compact [293] is the same as specifying --skip-add-drop-table [293], --skip-add-locks [293], --skip-comments [299], --skip-disable-keys [294], and --skip-set-charset [298] options.

To reverse the effect of a group option, uses its --skip-*xxx* form (--skip-opt [299] or --skip-compact [293]). It is also possible to select only part of the effect of a group option by following it with options that enable or disable specific features. Here are some examples:

- To select the effect of --opt [297] except for some features, use the --skip option for each feature. To disable extended inserts and memory buffering, use --opt [297] --skip-extended-insert [294] --skip-quick [298]. (As of MySQL 4.1, --skip-extended-insert [294] --skip-quick [298] is sufficient because --opt [297] is on by default.)
- To reverse --opt [297] for all features except index disabling and table locking, use --skip-opt [299] --disable-keys [294] --lock-tables [295].

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, --disable-keys [294] --lock-tables [295] --skip-opt [299] would not have the intended effect; it is the same as --skip-opt [299] by itself.

mysqldump can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the --quick [298] option (or --opt [297], which enables --quick [298]). The --opt [297] option (and hence --quick [298]) is enabled by default as of MySQL 4.1, so to enable memory buffering, use --skip-quick [298].

If you are using a recent version of mysqldump to generate a dump to be reloaded into a very old MySQL server, you should not use the --opt [297] or --extended-insert [294] option. Use --skip-opt [299] instead.

Before MySQL 4.1.2, out-of-range numeric values such as -inf and inf, as well as NaN (not-a-number) values are dumped by mysqldump as NULL. You can see this using the following sample table:

```
mysql> CREATE TABLE t (f DOUBLE);
mysql> INSERT INTO t VALUES(1e+11111111111111111111);
mysql> INSERT INTO t VALUES(-1e11111111111111111111);
mysql> SELECT f FROM t;
+------+
| f    |
+------+
|  inf |
| -inf |
+------+
```

For this table, mysqldump produces the following data output:

```
--
-- Dumping data for table `t`
--

INSERT INTO t VALUES (NULL);
INSERT INTO t VALUES (NULL);
```

The significance of this behavior is that if you dump and restore the table, the new table has contents that differ from the original contents. This problem is fixed as of MySQL 4.1.2; you cannot insert `inf` in the table, so this `mysqldump` behavior is only relevant when you deal with old servers.

For additional information about `mysqldump`, see Section 6.4, “Using mysqldump for Backups”.

`mysqldump` supports the following options, which can be specified on the command line or in the `[mysqldump]` and `[client]` option file groups. `mysqldump` also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.5 `mysqldump` Options

Format	Option File	Description	Introduced	Deprecated
--add-drop-database [292]	add-drop-database [292]	Add a DROP DATABASE statement before each CREATE DATABASE statement	4.1.13	
--add-drop-table [293]	add-drop-table [293]	Add a DROP TABLE statement before each CREATE TABLE statement		
--add-locks [293]	add-locks [293]	Surround each table dump with LOCK TABLES and UNLOCK TABLES statements		
--all-databases [293]	all-databases [293]	Dump all tables in all databases		
--allow-keywords [293]	allow-keywords [293]	Allow creation of column names that are keywords		
--comments [293]	comments [293]	Add comments to the dump file		
--compact [293]	compact [293]	Produce more compact output		
--compatible=name[,name,...] [293]	compatible [293]	Produce output that is more compatible with other database systems or with older MySQL servers		
--complete-insert [293]	complete-insert [293]	Use complete INSERT statements that include column names		
--create-options [294]	create-options [294]	Include all MySQL-specific table options in CREATE TABLE statements		
--databases [294]	databases [294]	Dump several databases		
--debug[=debug_options] [294]	debug [294]	Write a debugging log		
--default-character-set=charset_name [294]	default-character-set [294]	Use charset_name as the default character set		
--delayed-insert [294]	delayed-insert [294]	Write INSERT DELAYED statements rather than INSERT statements		

Format	Option File	Description	Introduced	Deprecated
--delete-master-logs [294]	delete-master-logs [294]	On a master replication server, delete the binary logs after performing the dump operation		
--disable-keys [294]	disable-keys [294]	For each table, surround the INSERT statements with statements to disable and enable keys		
--extended-insert [294]	extended-insert [294]	Use multiple-row INSERT syntax that include several VALUES lists		
--fields-enclosed-by=string [294]	fields-enclosed-by [294]	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--fields-escaped-by [294]	fields-escaped-by [294]	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--fields-optionally-enclosed-by=string [294]	fields-optionally-enclosed-by [294]	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--fields-terminated-by=string [294]	fields-terminated-by [294]	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--first-slave [294]	first-slave [294]	Deprecated; use --lock-all-tables instead		4.1.0
--flush-logs [294]	flush-logs [294]	Flush the MySQL server log files before starting the dump		
--help [292]		Display help message and exit		
--hex-blob [295]	hex-blob [295]	Dump binary columns using hexadecimal notation (for example, 'abc' becomes 0x616263)		
--host [295]	host [295]	Host to connect to (IP address or hostname)		
--ignore-table=db_name.tbl_name [295]	ignore-table [295]	Do not dump the given table		
--insert-ignore [295]	insert-ignore [295]	Write INSERT IGNORE statements rather than INSERT statements	4.1.12	
--lines-terminated-by=string [295]	lines-terminated-by [295]	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE		
--lock-all-tables [295]	lock-all-tables [295]	Lock all tables across all databases		
--lock-tables [295]	lock-tables [295]	Lock all tables before dumping them		
--master-data[=value] [296]	master-data [296]	Write the binary log file name and position to the output		
--max_allowed_packet=value [300]	max_allowed_packet [300]	The maximum packet length to send to or receive from the server		

Format	Option File	Description	Introduced	Deprecated
--net_buffer_length=value [300]	net_buffer_length [300]	The buffer size for TCP/IP and socket communication		
--no-autocommit [297]	no-autocommit [297]	Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements		
--no-create-db [297]	no-create-db [297]	This option suppresses the CREATE DATABASE statements		
--no-create-info [297]	no-create-info [297]	Do not write CREATE TABLE statements that re-create each dumped table		
--no-data [297]	no-data [297]	Do not dump table contents		
--no-set-names [297]	no-set-names [297]	Same as --skip-set-charset		4.1.0
--opt [297]	opt [297]	Shorthand for --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset.		
--order-by-primary [297]	order-by-primary [297]	Dump each table's rows sorted by its primary key, or by its first unique index		
--password[=password] [297]	password [297]	The password to use when connecting to the server		
--pipe [297]		On Windows, connect to server using a named pipe		
--port=port_num [297]	port [297]	The TCP/IP port number to use for the connection		
--protocol=type [298]	protocol [298]	The connection protocol to use		
--quick [298]	quick [298]	Retrieve rows for a table from the server a row at a time		
--quote-names [298]	quote-names [298]	Quote identifiers within backtick characters		
--result-file=file [298]	result-file [298]	Direct output to a given file		
--set-charset [298]	set-charset [298]	Add SET NAMES default_character_set to output		
--single-transaction [298]	single-transaction [298]	This option issues a BEGIN SQL statement before dumping data from the server		
--skip-add-drop-table [293]	skip-add-drop-table [293]	Do not add a DROP TABLE statement before each CREATE TABLE statement		
--skip-add-locks [293]	skip-add-locks [293]	Do not add locks		
--skip-comments [299]	skip-comments [299]	Do not add comments to the dump file		
--skip-compact [293]	skip-compact [293]	Do not produce more compact output		

Format	Option File	Description	Introduced	Deprecated
--skip-disable-keys [294]	skip-disable-keys [294]	Do not disable keys		
--skip-extended-insert [294]	skip-extended-insert [294]	Turn off extended-insert		
--skip-opt [299]	skip-opt [299]	Turn off the options set by --opt		
--skip-quick [298]	skip-quick [298]	Do not retrieve rows for a table from the server a row at a time		
--skip-quote-names [298]	skip-quote-names [298]	Do not quote identifiers		
--skip-set-charset [298]	skip-set-charset [298]	Suppress the SET NAMES statement		
--socket=path [299]	socket [299]	For connections to localhost		
--ssl-ca=file_name [299]	ssl-ca [299]	The path to a file that contains a list of trusted SSL CAs		
--ssl-capath=dir_name [299]	ssl-capath [299]	The path to a directory that contains trusted SSL CA certificates in PEM format		
--ssl-cert=file_name [299]	ssl-cert [299]	The name of the SSL certificate file to use for establishing a secure connection		
--ssl-cipher=cipher_list [299]	ssl-cipher [299]	A list of allowable ciphers to use for SSL encryption		
--ssl-key=file_name [299]	ssl-key [299]	The name of the SSL key file to use for establishing a secure connection		
--ssl-verify-server-cert [299]	ssl-verify-server-cert [299]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server		
--tab=path [299]	tab [299]	Produce tab-separated data files		
--tables [299]	tables [299]	Override the --databases or -B option		
--user=user_name [299]	user [299]	MySQL user name to use when connecting to server		
--verbose [299]		Verbose mode		
--version [299]		Display version information and exit		
--where='where_condition' [300]	where [300]	Dump only rows selected by the given WHERE condition		
--xml [300]	xml [300]	Produce XML output		

- `--help` [292], `-?`

  Display a help message and exit.

- `--add-drop-database` [292]

  Add a `DROP DATABASE` statement before each `CREATE DATABASE` statement. Added in MySQL 4.1.13.

- --add-drop-table [293]

  Add a DROP TABLE statement before each CREATE TABLE statement. This option is typically used in conjunction with the --all-databases [293] or --databases [294] option because no CREATE DATABASE statements are written unless one of those options is specified.

- --add-locks [293]

  Surround each table dump with LOCK TABLES and UNLOCK TABLES statements. This results in faster inserts when the dump file is reloaded. See Section 7.3.2.1, “Speed of INSERT Statements”.

- --all-databases [293], -A

  Dump all tables in all databases. This is the same as using the --databases [294] option and naming all the databases on the command line.

- --allow-keywords [293]

  Permit creation of column names that are keywords. This works by prefixing each column name with the table name.

- --character-sets-dir=*path* [293]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- --comments [293], -i

  Write additional information in the dump file such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use --skip-comments [299]. This option was added in MySQL 4.0.17.

- --compact [293]

  Produce more compact output. This option enables the --skip-add-drop-table [293], --skip-add-locks [293], --skip-comments [299], --skip-disable-keys [294], and --skip-set-charset [298] options. Added in MySQL 4.1.2.

- --compatible=*name* [293]

  Produce output that is more compatible with other database systems or with older MySQL servers. The value of *name* can be ansi, mysql323, mysql40, postgresql, oracle, mssql, db2, maxdb, no_key_options, no_table_options, or no_field_options. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See Section 5.1.6, “Server SQL Modes”.

  This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, --compatible=oracle [293] does not map data types to Oracle types or use Oracle comment syntax.

  This option requires a server version of 4.1.0 or higher. With older servers, it does nothing.

- --complete-insert [293], -c

  Use complete INSERT statements that include column names.

- --compress [293], -C

  Compress all information sent between the client and the server if both support compression.

- --create-options [294]

  Include all MySQL-specific table options in the CREATE TABLE statements. Before MySQL 4.1.2, use --all instead.

- --databases [294], -B

  Dump several databases. Normally, mysqldump treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. CREATE DATABASE and USE statements are included in the output before each new database.

- --debug[=*debug_options*] [294], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is 'd:t:o,*file_name*'. The default value is 'd:t:o,/tmp/mysqldump.trace'.

- --default-character-set=*charset_name* [294]

  Use *charset_name* as the default character set. See Section 9.6, “Character Set Configuration”. If no character set is specified, mysqldump from MySQL 4.1.2 or later uses utf8, and earlier versions use latin1.

- --delayed-insert [294]

  Write INSERT DELAYED statements rather than INSERT statements.

- --delete-master-logs [294]

  On a master replication server, delete the binary logs by sending a RESET MASTER statement to the server after performing the dump operation. This option automatically enables --first-slave [294] before MySQL 4.1.8 and enables --master-data [296] thereafter. It was added in MySQL 3.23.57 (for MySQL 3.23) and MySQL 4.0.13 (for MySQL 4.0).

- --disable-keys [294], -K

  For each table, surround the INSERT statements with /*!40000 ALTER TABLE *tbl_name* DISABLE KEYS */; and /*!40000 ALTER TABLE *tbl_name* ENABLE KEYS */; statements. This makes loading the dump file into a MySQL 4.0 or newer server faster because the indexes are created after all rows are inserted. This option is effective only for nonunique indexes of MyISAM tables. only.

- --extended-insert [294], -e

  Use multiple-row INSERT syntax that include several VALUES lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- --fields-terminated-by=... [294], --fields-enclosed-by=... [294], --fields-optionally-enclosed-by=... [294], --fields-escaped-by=... [294]

  These options are used with the --tab [299] option and have the same meaning as the corresponding FIELDS clauses for LOAD DATA INFILE. See Section 12.2.5, “LOAD DATA INFILE Syntax”.

- --first-slave [294]

  Deprecated. Use --lock-all-tables [295] instead as of MySQL 4.1.8.

- --flush-logs [294], -F

Flush the MySQL server log files before starting the dump. This option requires the RELOAD [492] privilege. If you use this option in combination with the --all-databases [293] option, the logs are flushed *for each database dumped*. The exception is when using --lock-all-tables [295] or --master-data [296]: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked. If you want your dump and the log flush to happen at exactly the same moment, you should use --flush-logs [294] together with either --lock-all-tables [295] or --master-data [296].

- --force [295], -f

  Continue even if an SQL error occurs during a table dump.

- --host=*host_name* [295], -h *host_name*

  Dump data from the MySQL server on the given host. The default host is localhost.

- --hex-blob [295]

  Dump binary columns using hexadecimal notation (for example, 'abc' becomes 0x616263). The affected data types are BINARY, VARBINARY, and the BLOB types in MySQL 4.1 and up, and CHAR BINARY, VARCHAR BINARY, and BLOB in MySQL 4.0. This option was added in MySQL 4.0.23 and 4.1.8.

- --ignore-table=*db_name.tbl_name* [295]

  Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times. This option was added in MySQL 4.1.9.

- --insert-ignore [295]

  Write INSERT IGNORE statements rather than INSERT statements. This option was added in MySQL 4.1.12.

- --lines-terminated-by=... [295]

  This option is used with the --tab [299] option and has the same meaning as the corresponding LINES clause for LOAD DATA INFILE. See Section 12.2.5, “LOAD DATA INFILE Syntax”.

- --lock-all-tables [295], -x

  Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off --single-transaction [298] and --lock-tables [295]. Added in MySQL 4.1.8.

- --lock-tables [295], -l

  For each dumped database, lock all tables to be dumped before dumping them. The tables are locked with READ LOCAL to permit concurrent inserts in the case of MyISAM tables. For transactional tables such as InnoDB and BDB, --single-transaction [298] is a much better option than --lock-tables [295] because it does not need to lock the tables at all.

  Because --lock-tables [295] locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

  This option has no effect for output data files produced by using the --tab [299] option. See the description for that option.

- `--master-data[=value]` [296]

  Use this option to dump a master replication server to produce a dump file that can be used to set up another server as a slave of the master. It causes the dump output to include a `CHANGE MASTER TO` statement that indicates the binary log coordinates (file name and position) of the dumped server. These are the master server coordinates from which the slave should start replicating after you load the dump file into the slave.

  If the option value is 2, the `CHANGE MASTER TO` statement is written as an SQL comment, and thus is informative only; it has no effect when the dump file is reloaded. If the option value is 1, the statement is not written as a comment and takes effect when the dump file is reloaded. If no option value is specified, the default value is 1. The value may be given as of MySQL 4.1.8; before that, do not specify an option value.

  This option requires the `RELOAD` [492] privilege and the binary log must be enabled.

  The `--master-data` [296] option automatically turns off `--lock-tables` [295]. It also turns on `--lock-all-tables` [295], unless `--single-transaction` [298] also is specified, in which case, a global read lock is acquired only for a short time at the beginning of the dump (see the description for `--single-transaction` [298]). In all cases, any action on logs happens at the exact moment of the dump.

  It is also possible to set up a slave by dumping an existing slave of the master. To do this, use the following procedure on the existing slave:

  1. Stop the slave's SQL thread and get its current status:

     ```
     mysql> STOP SLAVE SQL_THREAD;
     mysql> SHOW SLAVE STATUS;
     ```

  2. From the output of the `SHOW SLAVE STATUS` statement, the binary log coordinates of the master server from which the new slave should start replicating are the values of the `Relay_Master_Log_File` and `Exec_Master_Log_Pos` fields. Denote those values as *`file_name`* and *`file_pos`*.

  3. Dump the slave server:

     ```
     shell> mysqldump --master-data=2 --all-databases > dumpfile
     ```

  4. Restart the slave:

     ```
     mysql> START SLAVE;
     ```

  5. On the new slave, load the dump file:

     ```
     shell> mysql < dumpfile
     ```

  6. On the new slave, set the replication coordinates to those of the master server obtained earlier:

     ```
     mysql> CHANGE MASTER TO
         -> MASTER_LOG_FILE = 'file_name', MASTER_LOG_POS = file_pos;
     ```

     The `CHANGE MASTER TO` statement might also need other parameters, such as `MASTER_HOST` to point the slave to the correct master server host. Add any such parameters as necessary.

- `--no-autocommit` [297]

  Enclose the `INSERT` statements for each dumped table within `SET autocommit = 0` and `COMMIT` statements.

- `--no-create-db` [297], `-n`

  This option suppresses the `CREATE DATABASE` statements that are otherwise included in the output if the `--databases` [294] or `--all-databases` [293] option is given.

- `--no-create-info` [297], `-t`

  Do not write `CREATE TABLE` statements that re-create each dumped table.

- `--no-data` [297], `-d`

  Do not write any table row information (that is, do not dump table contents). This is useful if you want to dump only the `CREATE TABLE` statement for the table (for example, to create an empty copy of the table by loading the dump file).

- `--no-set-names` [297], `-N`

  This has the same effect as `--skip-set-charset` [298].

- `--opt` [297]

  This option is shorthand. It is the same as specifying `--add-drop-table` [293] `--add-locks` [293] `--create-options` [294] `--disable-keys` [294] `--extended-insert` [294] `--lock-tables` [295] `--quick` [298] `--set-charset` [298]. It should give you a fast dump operation and produce a dump file that can be reloaded into a MySQL server quickly.

  As of MySQL 4.1, `--opt` [297] is enabled by default. Use `--skip-opt` [299] to disable it. See the discussion at the beginning of this section for information about selectively enabling or disabling a subset of the options affected by `--opt` [297].

- `--order-by-primary` [297]

  Dump each table's rows sorted by its primary key, or by its first unique index, if such an index exists. This is useful when dumping a `MyISAM` table to be loaded into an `InnoDB` table, but will make the dump operation take considerably longer. This option was added in MySQL 4.1.8.

- `--password[=`*`password`*`]` [297], `-p[`*`password`*`]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` [297] or `-p` option on the command line, `mysqldump` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe` [297], `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=`*`port_num`* [297], `-P` *`port_num`*

  The TCP/IP port number to use for the connection.

- --protocol={TCP|SOCKET|PIPE|MEMORY} [298]

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”. This option was added in MySQL 4.1.

- --quick [298], -q

  This option is useful for dumping large tables. It forces mysqldump to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- --quote-names [298], -Q

  Quote identifiers (such as database, table, and column names) within “`” characters. If the ANSI_QUOTES [458] SQL mode is enabled, identifiers are quoted within “"” characters. As of MySQL 4.1.1, --quote-names [298] is enabled by default. It can be disabled with --skip-quote-names [298], but this option should be given after any option such as --compatible [293] that may enable --quote-names [298].

- --result-file=*file_name* [298], -r *file_name*

  Direct output to a given file. This option should be used on Windows to prevent newline “\n” characters from being converted to “\r\n” carriage return/newline sequences. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.

- --set-charset [298]

  Add SET NAMES *default_character_set* to the output. This option is enabled by default. To suppress the SET NAMES statement, use --skip-set-charset [298]. This option was added in MySQL 4.1.2.

- --single-transaction [298]

  This option sends a START TRANSACTION SQL statement to the server before dumping data. It is useful only with transactional tables such as InnoDB and BDB, because then it dumps the consistent state of the database at the time when BEGIN was issued without blocking any applications.

  When using this option, you should keep in mind that only InnoDB tables are dumped in a consistent state. For example, any MyISAM or MEMORY tables dumped while using this option may still change state.

  While a --single-transaction [298] dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: ALTER TABLE, CREATE TABLE, DROP TABLE, RENAME TABLE, TRUNCATE TABLE. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the SELECT that is performed by mysqldump to retrieve the table contents to obtain incorrect contents or fail.

  The --single-transaction [298] option was added in MySQL 4.0.2. This option is mutually exclusive with the --lock-tables [295] option because LOCK TABLES causes any pending transactions to be committed implicitly.

  This option is not supported for MySQL Cluster tables; the results cannot be guaranteed to be consistent due to the fact that the NDBCLUSTER storage engine supports only the READ_COMMITTED transaction isolation level. You should always use NDB backup and restore instead.

To dump large tables, combine the --single-transaction [298] option with the --quick [298] option.

- --skip-comments [299]

  See the description for the --comments [293] option.

- --skip-opt [299]

  See the description for the --opt [297] option.

- --socket=*path* [299], -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

  Options that begin with --ssl [521] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, “SSL Command Options”.

- --tab=*path* [299], -T *path*

  Produce tab-separated text-format data files. For each dumped table, mysqldump creates a *tbl_name*.sql file that contains the CREATE TABLE statement that creates the table, and the server writes a *tbl_name*.txt file that contains its data. The option value is the directory in which to write the files.

  Note

  This option should be used only when mysqldump is run on the same machine as the mysqld server. You must have the FILE [491] privilege, and the server must have permission to write files in the directory that you specify.

  By default, the .txt data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the --fields-*xxx* and --lines-terminated-by [295] options.

  Column values are dumped using the binary character set and the --default-character-set [294] option is ignored. In effect, there is no character set conversion. If a table contains columns in several character sets, the output data file will as well and you may not be able to reload the file correctly.

- --tables [299]

  Override the --databases [294] or -B option. mysqldump regards all name arguments following the option as table names.

- --user=*user_name* [299], -u *user_name*

  The MySQL user name to use when connecting to the server.

- --verbose [299], -v

  Verbose mode. Print more information about what the program does.

- --version [299], -V

Display version information and exit.

- --where='*where_condition*' [300], -w '*where_condition*'

  Dump only rows selected by the given WHERE condition. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

  Examples:

```
--where="user='jimf'"
-w"userid>1"
-w"userid<1"
```

- --xml [300], -X

  Write dump output as well-formed XML.

You can also set the following variables by using --*var_name*=*value* syntax:

- max_allowed_packet

  The maximum size of the buffer for client/server communication. The default is 24MB. The maximum can be up to 16MB before MySQL 4.0, and up to 1GB from MySQL 4.0 on.

- net_buffer_length [422]

  The initial size of the buffer for client/server communication. When creating multiple-row INSERT statements (as with the --extended-insert [294] or --opt [297] option), mysqldump creates rows up to net_buffer_length [422] length. If you increase this variable, you should also ensure that the net_buffer_length [422] variable in the MySQL server is at least this large.

It is also possible to set variables by using --set-variable=*var_name*=*value* or -O *var_name*=*value* syntax. However, this syntax is deprecated as of MySQL 4.0.

A common use of mysqldump is for making a backup of an entire database:

```
shell> mysqldump db_name > backup-file.sql
```

You can load the dump file back into the server like this:

```
shell> mysql db_name < backup-file.sql
```

Or like this:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

mysqldump is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

It is possible to dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the --all-databases [293] option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For InnoDB tables, mysqldump provides a way of making an online backup:

```
shell> mysqldump --all-databases --single-transaction > all_databases.sql
```

This backup acquires a global read lock on all tables (using FLUSH TABLES WITH READ LOCK) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the MySQL server may get stalled until those statements finish. After that, the dump becomes lock free and does not disturb reads and writes on the tables. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see Section 5.3.4, “The Binary Log”) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2
            > all_databases.sql
```

The --master-data [296] and --single-transaction [298] options can be used simultaneously as of MySQL 4.1.8, which provides a convenient way to make an online backup suitable for use prior to point-in-time recovery if tables are stored using the InnoDB storage engine.

For more information on making backups, see Section 6.2, “Database Backup Methods”, and Section 6.3, “Example Backup and Recovery Strategy”.

4.5.5 mysqlimport — A Data Import Program

The mysqlimport client provides a command-line interface to the LOAD DATA INFILE SQL statement. Most options to mysqlimport correspond directly to clauses of LOAD DATA INFILE syntax. See Section 12.2.5, “LOAD DATA INFILE Syntax”.

Invoke mysqlimport like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, mysqlimport strips any extension from the file name and uses the result to determine the name of the table into which to import the file's contents. For example, files named patient.txt, patient.text, and patient all would be imported into a table named patient.

mysqlimport supports the following options, which can be specified on the command line or in the [mysqlimport] and [client] option file groups. mysqlimport also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.6 mysqlimport Options

Format	Option File	Description
--columns=column_list [303]	columns [303]	This option takes a comma-separated list of column names as its value
--compress [303]	compress [303]	Compress all information sent between the client and the server
--debug[=debug_options] [303]	debug [303]	Write a debugging log
--default-character-set=charset_name [303]	default-character-set [303]	Use charset_name as the default character set
--delete [303]	delete [303]	Empty the table before importing the text file
--fields-enclosed-by=string [303]	fields-enclosed-by [303]	This option has the same meaning as the corresponding clause for LOAD DATA INFILE
--fields-escaped-by [303]	fields-escaped-by [303]	This option has the same meaning as the corresponding clause for LOAD DATA INFILE
--fields-optionally-enclosed-by=string [303]	fields-optionally-enclosed-by [303]	This option has the same meaning as the corresponding clause for LOAD DATA INFILE
--fields-terminated-by=string [303]	fields-terminated-by [303]	-- This option has the same meaning as the corresponding clause for LOAD DATA INFILE
--force [304]	force [304]	Continue even if an SQL error occurs
--help [303]		Display help message and exit
--host=host_name [304]	host [304]	Connect to the MySQL server on the given host
--ignore [304]	ignore [304]	See the description for the --replace option
--ignore-lines=# [304]	ignore-lines [304]	Ignore the first N lines of the data file
--lines-terminated-by=string [304]	lines-terminated-by [304]	This option has the same meaning as the corresponding clause for LOAD DATA INFILE
--local [304]	local [304]	Read input files locally from the client host
--lock-tables [304]	lock-tables [304]	Lock all tables for writing before processing any text files
--low-priority [304]	low-priority [304]	Use LOW_PRIORITY when loading the table.
--password[=password] [304]	password [304]	The password to use when connecting to the server
--pipe [304]		On Windows, connect to server using a named pipe
--port=port_num [304]	port [304]	The TCP/IP port number to use for the connection
--protocol=type [304]	protocol [304]	The connection protocol to use
--replace [305]	replace [305]	The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values
--silent [305]	silent [305]	Produce output only when errors occur
--socket=path [305]	socket [305]	For connections to localhost

Format	Option File	Description
--ssl-ca=file_name [305]	ssl-ca [305]	The path to a file that contains a list of trusted SSL CAs
--ssl-capath=dir_name [305]	ssl-capath [305]	The path to a directory that contains trusted SSL CA certificates in PEM format
--ssl-cert=file_name [305]	ssl-cert [305]	The name of the SSL certificate file to use for establishing a secure connection
--ssl-cipher=cipher_list [305]	ssl-cipher [305]	A list of allowable ciphers to use for SSL encryption
--ssl-key=file_name [305]	ssl-key [305]	The name of the SSL key file to use for establishing a secure connection
--ssl-verify-server-cert [305]	ssl-verify-server-cert [305]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server
--user=user_name, [305]	user [305]	MySQL user name to use when connecting to server
--verbose [305]		Verbose mode
--version [305]		Display version information and exit

- `--help` [303], `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`path`* [303]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--columns=`*`column_list`* [303], `-c` *`column_list`*

  This option takes a comma-separated list of column names as its value. The order of the column names indicates how to match data file columns with table columns.

- `--compress` [303], `-C`

  Compress all information sent between the client and the server if both support compression.

- `--debug[=`*`debug_options`*`]` [303], `-#` `[`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:o,`*`file_name`*`'`. The default is `'d:t:o'`.

- `--default-character-set=`*`charset_name`* [303]

  Use *`charset_name`* as the default character set. See Section 9.6, “Character Set Configuration”.

- `--delete` [303], `-D`

  Empty the table before importing the text file.

- `--fields-terminated-by=...` [303], `--fields-enclosed-by=...` [303], `--fields-optionally-enclosed-by=...` [303], `--fields-escaped-by=...` [303]

  These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See Section 12.2.5, “`LOAD DATA INFILE` Syntax”.

- `--force` [304], `-f`

  Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without `--force` [304], `mysqlimport` exits if a table does not exist.

- `--host=host_name` [304], `-h host_name`

  Import data to the MySQL server on the given host. The default host is `localhost`.

- `--ignore` [304], `-i`

  See the description for the `--replace` [305] option.

- `--ignore-lines=N` [304]

  Ignore the first `N` lines of the data file.

- `--lines-terminated-by=...` [304]

  This option has the same meaning as the corresponding clause for `LOAD DATA INFILE`. For example, to import Windows files that have lines terminated with carriage return/linefeed pairs, use `--lines-terminated-by="\r\n"` [304]. (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See Section 12.2.5, “`LOAD DATA INFILE` Syntax”.

- `--local` [304], `-L`

  Read input files locally from the client host.

- `--lock-tables` [295], `-l`

  Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--low-priority` [304]

  Use `LOW_PRIORITY` when loading the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- `--password[=password]` [304], `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` [304] or `-p` option on the command line, `mysqlimport` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe` [304], `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=port_num` [304], `-P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [304]

The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”. This option was added in MySQL 4.1.

- `--replace` [305], `-r`

  The `--replace` [305] and `--ignore` [304] options control handling of input rows that duplicate existing rows on unique key values. If you specify `--replace` [305], new rows replace existing rows that have the same unique key value. If you specify `--ignore` [304], input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--silent` [305], `-s`

  Silent mode. Produce output only when errors occur.

- `--socket=`*`path`* [305], `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` [521] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, “SSL Command Options”.

- `--user=`*`user_name`* [305], `-u` *`user_name`*

  The MySQL user name to use when connecting to the server.

- `--verbose` [305], `-v`

  Verbose mode. Print more information about what the program does.

- `--version` [305], `-V`

  Display version information and exit.

Here is a sample session that demonstrates use of `mysqlimport`:

```
shell> mysql -e 'CREATE TABLE imptest(id INT, n VARCHAR(30))' test
shell> ed
a
100     Max Sydow
101     Count Dracula
.
w imptest.txt
32
q
shell> od -c imptest.txt
0000000   1   0   0  \t   M   a   x       S   y   d   o   w  \n   1   0
0000020   1  \t   C   o   u   n   t       D   r   a   c   u   l   a  \n
0000040
shell> mysqlimport --local test imptest.txt
test.imptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
shell> mysql -e 'SELECT * FROM imptest' test
+------+---------------+
| id   | n             |
+------+---------------+
```

```
|  100 | Max Sydow     |
|  101 | Count Dracula |
+------+---------------+
```

4.5.6 mysqlshow — Display Database, Table, and Column Information

The mysqlshow client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

mysqlshow provides a command-line interface to several SQL SHOW statements. See Section 12.4.5, “SHOW Syntax”. The same information can be obtained by using those statements directly. For example, you can issue them from the mysql client program.

Invoke mysqlshow like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (“*”, “?”, “%”, or “_”), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. “*” and “?” characters are converted into SQL “%” and “_” wildcard characters. This might cause some confusion when you try to display the columns for a table with a “_” in the name, because in this case, mysqlshow shows you only the table names that match the pattern. This is easily fixed by adding an extra “%” last on the command line as a separate argument.

mysqlshow supports the following options, which can be specified on the command line or in the [mysqlshow] and [client] option file groups. mysqlshow also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.7 mysqlshow Options

Format	Option File	Description
--compress [307]	compress [307]	Compress all information sent between the client and the server
--debug[=debug_options] [307]	debug [307]	Write a debugging log
--default-character-set=charset_name [307]	default-character-set [307]	Use charset_name as the default character set
--help [307]		Display help message and exit
--host=host_name [307]	host [307]	Connect to the MySQL server on the given host
--keys [307]	keys [307]	Show table indexes
--password[=password] [308]	password [308]	The password to use when connecting to the server
--pipe [308]		On Windows, connect to server using a named pipe

Format	Option File	Description
--port=port_num [308]	port [308]	The TCP/IP port number to use for the connection
--protocol=type [308]	protocol [308]	The connection protocol to use
--socket=path [308]	socket [308]	For connections to localhost
--ssl-ca=file_name [308]	ssl-ca [308]	The path to a file that contains a list of trusted SSL CAs
--ssl-capath=dir_name [308]	ssl-capath [308]	The path to a directory that contains trusted SSL CA certificates in PEM format
--ssl-cert=file_name [308]	ssl-cert [308]	The name of the SSL certificate file to use for establishing a secure connection
--ssl-cipher=cipher_list [308]	ssl-cipher [308]	A list of allowable ciphers to use for SSL encryption
--ssl-key=file_name [308]	ssl-key [308]	The name of the SSL key file to use for establishing a secure connection
--ssl-verify-server-cert [308]	ssl-verify-server-cert [308]	The server's Common Name value in its certificate is verified against the host name used when connecting to the server
--status [308]	status [308]	Display extra information about each table
--user=user_name, [308]	user [308]	MySQL user name to use when connecting to server
--verbose [308]		Verbose mode
--version [308]		Display version information and exit

- `--help` [307], `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`path`* [307]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--compress` [307], `-C`

  Compress all information sent between the client and the server if both support compression.

- `--debug[=`*`debug_options`*`]` [307], `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:o,`*`file_name`*`'`. The default is `'d:t:o'`.

- `--default-character-set=`*`charset_name`* [307]

  Use *`charset_name`* as the default character set. See Section 9.6, “Character Set Configuration”.

- `--host=`*`host_name`* [307], `-h` *`host_name`*

  Connect to the MySQL server on the given host.

- `--keys` [307], `-k`

Show table indexes.

- `--password[=password]` [308], `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` [308] or `-p` option on the command line, `mysqlshow` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line. You can use an option file to avoid giving the password on the command line.

- `--pipe` [308], `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--port=port_num` [308], `-P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [308]

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”. This option was added in MySQL 4.1.

- `--socket=path` [308], `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` [521] specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, “SSL Command Options”.

- `--status` [308], `-i`

  Display extra information about each table.

- `--user=user_name` [308], `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose` [308], `-v`

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version` [308], `-V`

  Display version information and exit.

  Some options, such as `--opt` [297], automatically enable `--lock-tables`. If you want to override this, use `--skip-lock-tables` at the end of the option list.

4.6 MySQL Administrative and Utility Programs

This section describes administrative programs and programs that perform miscellaneous utility operations.

4.6.1 `myisam_ftdump` — Display Full-Text Index information

`myisam_ftdump` displays information about `FULLTEXT` indexes in `MyISAM` tables. It reads the `MyISAM` index file directly, so it must be run on the server host where the table is located. Before using `myisam_ftdump`, be sure to issue a `FLUSH TABLES` statement first if the server is running.

`myisam_ftdump` scans and dumps the entire index, which is not particularly fast. On the other hand, the distribution of words changes infrequently, so it need not be run often.

Invoke `myisam_ftdump` like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The *`tbl_name`* argument should be the name of a `MyISAM` table. You can also specify a table by naming its index file (the file with the `.MYI` suffix). If you do not invoke `myisam_ftdump` in the directory where the table files are located, the table or index file name must be preceded by the path name to the table's database directory. Index numbers begin with 0.

Example: Suppose that the `test` database contains a table named `mytexttable` that has the following definition:

```
CREATE TABLE mytexttable
(
  id   INT NOT NULL,
  txt  TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
) ENGINE=MyISAM;
```

The index on `id` is index 0 and the `FULLTEXT` index on `txt` is index 1. If your working directory is the `test` database directory, invoke `myisam_ftdump` as follows:

```
shell> myisam_ftdump mytexttable 1
```

If the path name to the `test` database directory is `/usr/local/mysql/data/test`, you can also specify the table name argument using that path name. This is useful if you do not invoke `myisam_ftdump` in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

You can use `myisam_ftdump` to generate a list of index entries in order of frequency of occurrence like this:

```
shell> myisam_ftdump -c mytexttable 1 | sort -r
```

`myisam_ftdump` supports the following options:

- `--help` [309], `-h` `-?`

  Display a help message and exit.

- `--count` [310], `-c`

  Calculate per-word statistics (counts and global weights).

- `--dump` [310], `-d`

  Dump the index, including data offsets and word weights.

- `--length` [310], `-l`

  Report the length distribution.

- `--stats` [310], `-s`

  Report global index statistics. This is the default operation if no other operation is specified.

- `--verbose` [310], `-v`

  Verbose mode. Print more output about what the program does.

4.6.2 myisamchk — MyISAM Table-Maintenance Utility

The `myisamchk` utility gets information about your database tables or checks, repairs, or optimizes them. `myisamchk` works with `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes). A related utility, `isamchk`, works with `ISAM` tables (tables that have `.ISD` and `.ISM` files for storing data and indexes).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 12.4.2.3, “`CHECK TABLE` Syntax”, and Section 12.4.2.6, “`REPAIR TABLE` Syntax”.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

Invoke `myisamchk` like this:

```
shell> myisamchk [options] tbl_name ...
```

The *`options`* specify what you want `myisamchk` to do. They are described in the following sections. You can also get a list of options by invoking `myisamchk --help`.

With no options, `myisamchk` simply checks your table as the default operation. To get more information or to tell `myisamchk` to take corrective action, specify options as described in the following discussion.

`tbl_name` is the database table you want to check or repair. If you run `myisamchk` somewhere other than in the database directory, you must specify the path to the database directory, because `myisamchk` has no idea where the database is located. In fact, `myisamchk` does not actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the `myisamchk` command line if you wish. You can also specify a table by naming its index file (the file with the `.MYI` suffix). This enables you to specify all tables in a directory by using the pattern `*.MYI`. For example, if you are in a database directory, you can check all the `MyISAM` tables in that directory like this:

```
```

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM and ISAM tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
shell> isamchk --silent /path/to/datadir/*/*.ISM
```

If you want to check all MyISAM and ISAM tables and repair any that are corrupted, you can use the following commands:

```
shell> myisamchk --silent --force --fast --update-state \
          --key_buffer_size=64M --sort_buffer_size=64M \
          --read_buffer_size=1M --write_buffer_size=1M \
          /path/to/datadir/*/*.MYI
shell> isamchk --silent --force --key_buffer_size=64M \
          --sort_buffer_size=64M --read_buffer_size=1M --write_buffer_size=1M \
          /path/to/datadir/*/*.ISM
```

These commands assume that you have more than 64MB free. For more information about memory allocation with myisamchk, see Section 4.6.2.6, “myisamchk Memory Usage”.

For additional information about using myisamchk, see Section 6.6, “MyISAM Table Maintenance and Crash Recovery”.

Important

You must ensure that no other program is using the tables while you are running myisamchk. The most effective means of doing so is to shut down the MySQL server while running myisamchk, or to lock all tables that myisamchk is being used on.

Otherwise, when you run myisamchk, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the mysqld server) that hasn't yet closed the file or that has died without closing the file properly, which can sometimes lead to the corruption of one or more MyISAM tables.

If mysqld is running, you must force it to flush any table modifications that are still buffered in memory by using FLUSH TABLES. You should then ensure that no one is using the tables while you are running myisamchk

However, the easiest way to avoid this problem is to use CHECK TABLE instead of myisamchk to check tables. See Section 12.4.2.3, “CHECK TABLE Syntax”.

myisamchk supports the following options, which can be specified on the command line or in the [myisamchk] option file group. myisamchk also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.8 myisamchk Options

Format	Option File	Description
--analyze [318]	analyze [318]	Analyze the distribution of key values
--backup [317]	backup [317]	Make a backup of the .MYD file as file_name-time.BAK
--block-search=offset [318]	block-search [318]	Find the record that a block at the given offset belongs to
--check [316]	check [316]	Check the table for errors
--check-only-changed [316]	check-only-changed [316]	Check only tables that have changed since the last check
--correct-checksum [317]	correct-checksum [317]	Correct the checksum information for the table
--data-file-length=len [317]	data-file-length [317]	Maximum length of the data file (when re-creating data file when it is full)
--debug[=debug_options] [314]	debug [314]	Write a debugging log
decode_bits=#	decode_bits	Decode_bits
--description [319]	description [319]	Print some descriptive information about the table
--extend-check [316]	extend-check [316]	Do very thorough table check or repair that tries to recover every possible row from the data file
--fast [316]	fast [316]	Check only tables that haven't been closed properly
--force [316]	force [316]	Do a repair operation automatically if myisamchk finds any errors in the table
--force	force-recover	Overwrite old temporary files. For use with the -r or -o option
ft_max_word_len=#	ft_max_word_len	Maximum word length for FULLTEXT indexes
ft_min_word_len=#	ft_min_word_len	Minimum word length for FULLTEXT indexes
ft_stopword_file=value	ft_stopword_file	Use stopwords from this file instead of built-in list
--HELP [313]		Display help message and exit
--help [313]		Display help message and exit
--information [316]	information [316]	Print informational statistics about the table that is checked
key_buffer_size=#	key_buffer_size	The size of the buffer used for index blocks for MyISAM tables
--keys-used=val [317]	keys-used [317]	A bit-value that indicates which indexes to update
--medium-check [316]	medium-check [316]	Do a check that is faster than an --extend-check operation
myisam_block_size=#	myisam_block_size	Block size to be used for MyISAM index pages
--parallel-recover [317]	parallel-recover [317]	Uses the same technique as -r and -n, but creates all the keys in parallel, using different threads (beta)

Format	Option File	Description
--quick [317]	quick [317]	Achieve a faster repair by not modifying the data file.
read_buffer_size=#	read_buffer_size	Each thread that does a sequential scan allocates a buffer of this size for each table it scans
--read-only [316]	read-only [316]	Don't mark the table as checked
--recover [317]	recover [317]	Do a repair that can fix almost any problem except unique keys that aren't unique
--safe-recover [318]	safe-recover [318]	Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found
--set-auto-increment[=value] [319]	set-auto-increment [319]	Force AUTO_INCREMENT numbering for new records to start at the given value
--set-character-set=name [318]	set-character-set [318]	Change the character set used by the table indexes
--set-collation=name [318]	set-collation [318]	Specify the collation to use for sorting table indexes
--silent [314]	silent [314]	Silent mode
sort_buffer_size=#	sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE
--sort-index [319]	sort-index [319]	Sort the index tree blocks in high-low order
sort_key_blocks=#	sort_key_blocks	sort_key_blocks
--sort-records=# [319]	sort-records [319]	Sort records according to a particular index
--sort-recover [318]	sort-recover [318]	Force myisamchk to use sorting to resolve the keys even if the temporary files would be very large
stats_method=value	stats_method	Specifies how MyISAM index statistics collection code should treat NULLs
--tmpdir=path [318]	tmpdir [318]	Path of the directory to be used for storing temporary files
--unpack [318]	unpack [318]	Unpack a table that was packed with myisampack
--update-state [316]	update-state [316]	Store information in the .MYI file to indicate when the table was checked and whether the table crashed
--verbose [314]		Verbose mode
--version [314]		Display version information and exit
write_buffer_size=#	write_buffer_size	Write buffer size

4.6.2.1 myisamchk General Options

The options described in this section can be used for any type of table maintenance operation performed by myisamchk. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- --help [313], -?

  Display a help message and exit. Options are grouped by type of operation.

- --HELP [313], -H

Display a help message and exit. Options are presented in a single list.

- `--debug=debug_options` [314], `-# debug_options`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/myisamchk.trace'`.

- `--silent` [314], `-s`

  Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make `myisamchk` very silent.

- `--verbose` [314], `-v`

  Verbose mode. Print more information about what the program does. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for even more output.

- `--version` [314], `-V`

  Display version information and exit.

- `--wait` [314], `-w`

  Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. If you are running `mysqld` with external locking disabled, the table can be locked only by another `myisamchk` command.

You can also set the following variables by using `--var_name=value` syntax:

Variable	Default Value
`decode_bits`	9
`ft_max_word_len`	version-dependent
`ft_min_word_len`	4
`ft_stopword_file`	built-in list
`key_buffer_size`	523264
`myisam_block_size`	1024
`read_buffer_size`	262136
`sort_buffer_size`	2097144
`sort_key_blocks`	16
`stats_method`	nulls_unequal
`write_buffer_size`	262136

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

The possible `myisamchk` variables and their default values can be examined with `myisamchk --help`:

`sort_buffer_size` is used when the keys are repaired by sorting keys, which is the normal case when you use `--recover` [317].

`key_buffer_size` is used when you are checking the table with `--extend-check` [316] or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use `--safe-recover` [318].
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for `CHAR`, `VARCHAR`, or `TEXT` columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force `myisamchk` to repair by sorting, you can use the `--sort-recover` [318] option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the `key_buffer_size` and `sort_buffer_size` variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

`myisam_block_size` is the size used for index blocks. It is available as of MySQL 4.0.0.

`stats_method` influences how `NULL` values are treated for index statistics collection when the `--analyze` [318] option is given. It acts like the `myisam_stats_method` system variable. For more information, see the description of `myisam_stats_method` in Section 5.1.3, “Server System Variables”, and Section 7.4.4, “`MyISAM` Index Statistics Collection”. `stats_method` was added in MySQL 4.1.15/5.0.14. For older versions, the statistics collection method is equivalent to `nulls_equal`.

The `ft_min_word_len` and `ft_max_word_len` variables are available as of MySQL 4.0.0. `ft_stopword_file` is available as of MySQL 4.0.19.

`ft_min_word_len` and `ft_max_word_len` indicate the minimum and maximum word length for `FULLTEXT` indexes. `ft_stopword_file` names the stopword file. These need to be set under the following circumstances.

If you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, you can place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE`. These statements are performed by the server, which knows the proper full-text parameter values to use.

4.6.2.2 `myisamchk` Check Options

myisamchk supports the following options for table checking operations:

- --check [316], -c

  Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- --check-only-changed [316], -C

  Check only tables that have changed since the last check.

- --extend-check [316], -e

  Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, myisamchk or myisamchk --medium-check should be able to determine whether there are any errors in the table.

  If you are using --extend-check [316] and have plenty of memory, setting the key_buffer_size variable to a large value helps the repair operation run faster.

  For a description of the output format, see Section 4.6.2.5, “Obtaining Table Information with myisamchk”.

- --fast [316], -F

  Check only tables that haven't been closed properly.

- --force [316], -f

  Do a repair operation automatically if myisamchk finds any errors in the table. The repair type is the same as that specified with the --recover [317] or -r option.

- --information [316], -i

  Print informational statistics about the table that is checked.

- --medium-check [316], -m

  Do a check that is faster than an --extend-check [316] operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- --read-only [316], -T

  Do not mark the table as checked. This is useful if you use myisamchk to check a table that is in use by some other application that does not use locking, such as mysqld when run with external locking disabled.

- --update-state [316], -U

  Store information in the .MYI file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the --check-only-changed [316] option, but you shouldn't use this option if the mysqld server is using the table and you are running it with external locking disabled.

4.6.2.3 myisamchk Repair Options

myisamchk supports the following options for table repair operations (operations performed when an option such as --recover [317] or --safe-recover [318] is given):

- `--backup` [317], `-B`

  Make a backup of the `.MYD` file as *`file_name-time`*`.BAK`

- `--character-sets-dir=`*`path`* [317]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--correct-checksum` [317]

  Correct the checksum information for the table.

- `--data-file-length=`*`len`* [317], `-D` *`len`*

  The maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check` [316], `-e`

  Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Do not use this option unless you are desperate.

  For a description of the output format, see Section 4.6.2.5, “Obtaining Table Information with `myisamchk`”.

- `--force` [316], `-f`

  Overwrite old intermediate files (files with names like *`tbl_name`*`.TMD`) instead of aborting.

- `--keys-used=`*`val`* [317], `-k` *`val`*

  For `myisamchk`, the option value is a bit-value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. For `isamchk`, the option value indicates that only the first *`val`* of the table indexes should be updated. In either case, an option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using `myisamchk -r` or (`isamchk -r`).

- `--no-symlinks` [317], `-l`

  Do not follow symbolic links. Normally `myisamchk` repairs the table that a symlink points to. This option does not exist as of MySQL 4.0 because versions from 4.0 on do not remove symlinks during repair operations.

- `--max-record-length=`*`len`* [317]

  Skip rows larger than the given length if `myisamchk` cannot allocate memory to hold them. This option was added in MySQL 4.1.1.

- `--parallel-recover` [317], `-p`

  Uses the same technique as `-r` and `-n`, but creates all the keys in parallel, using different threads. This option was added in MySQL 4.0.2. *This is beta-quality code; use at your own risk!*.

- `--quick` [317], `-q`

  Achieve a faster repair by modifying only the index file, not the data file. You can specify this option twice to force `myisamchk` to modify the original data file in case of duplicate keys.

- `--recover` [317], `-r`

Do a repair that can fix almost any problem except unique keys that are not unique (which is an extremely unlikely error with ISAM/MyISAM tables). If you want to recover a table, this is the option to try first. You should try --safe-recover [318] only if myisamchk reports that the table cannot be recovered by --recover [317]. (In the unlikely case that --recover [317] fails, the data file remains intact.)

If you have lots of memory, you should increase the value of sort_buffer_size.

- --safe-recover [318], -o

  Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than --recover [317], but can handle a couple of very unlikely cases that --recover [317] cannot. This recovery method also uses much less disk space than --recover [317]. Normally, you should repair first using --recover [317], and then with --safe-recover [318] only if --recover [317] fails.

  If you have lots of memory, you should increase the value of key_buffer_size.

- --set-character-set=*name* [318]

  Change the character set used by the table indexes. This option was replaced by --set-collation [318] in MySQL 4.1.1.

- --set-collation=*name* [318]

  Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name. This option was added in MySQL 4.1.11.

- --sort-recover [318], -n

  Force myisamchk to use sorting to resolve the keys even if the temporary files should be very large.

- --tmpdir=*path* [318], -t *path*

  The path of the directory to be used for storing temporary files. If this is not set, myisamchk uses the value of the TMPDIR environment variable. Starting from MySQL 4.1, --tmpdir [318] can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names should be colon (":") on Unix and semicolon (";") on Windows, NetWare, and OS/2.

- --unpack [318], -u

  Unpack a table that was packed with myisampack.

4.6.2.4 Other myisamchk Options

myisamchk supports the following options for actions other than table checks and repairs:

- --analyze [318], -a

  Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a myisamchk --description --verbose *tbl_name* command or the SHOW INDEX FROM *tbl_name* statement.

- --block-search=*offset* [318], -b *offset*

Find the record that a block at the given offset belongs to.

- `--description` [319], `-d`

  Print some descriptive information about the table. Specifying the `--verbose` [314] option once or twice produces additional information. See Section 4.6.2.5, “Obtaining Table Information with myisamchk”.

- `--set-auto-increment[=value]` [319], `-A[value]`

  Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If *`value`* is not specified, `AUTO_INCREMENT` numbers for new records begin with the largest value currently in the table, plus one.

- `--sort-index` [319], `-S`

  Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=N` [319], `-R N`

  Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that `myisamchk` sees them. Indexes are numbered beginning with 1.

  If keys are not packed (`PACK_KEYS=0`), they have the same length, so when `myisamchk` sorts and moves records, it just overwrites record offsets in the index. If keys are packed (`PACK_KEYS=1`), `myisamchk` must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

4.6.2.5 Obtaining Table Information with `myisamchk`

To obtain a description of a `MyISAM` table or statistics about it, use the commands shown here. The output from these commands is explained later in this section.

- `myisamchk -d tbl_name`

  Runs `myisamchk` in “describe mode” to produce a description of your table. If you start the MySQL server with external locking disabled, `myisamchk` may report an error for a table that is updated while it runs. However, because `myisamchk` does not change the table in describe mode, there is no risk of destroying data.

- `myisamchk -dv tbl_name`

  Adding `-v` runs `myisamchk` in verbose mode so that it produces more information about the table. Adding `-v` a second time produces even more information.

- `myisamchk -eis tbl_name`

  Shows only the most important information from a table. This operation is slow because it must read the entire table.

- `myisamchk -eiv tbl_name`

  This is like `-eis`, but tells you what is being done.

The *tbl_name* argument can be either the name of a MyISAM table or the name of its index file, as described in Section 4.6.2, “myisamchk — MyISAM Table-Maintenance Utility”. Multiple *tbl_name* arguments can be given.

Suppose that a table named person has the following structure. (The MAX_ROWS table option is included so that in the example output from myisamchk shown later, some values are smaller and fit the output format more easily.)

```
CREATE TABLE person
(
  id         INT NOT NULL AUTO_INCREMENT,
  last_name  VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  birth      DATE,
  death      DATE,
  PRIMARY KEY (id),
  INDEX (last_name, first_name),
  INDEX (birth)
) MAX_ROWS = 1000000;
```

Suppose also that the table has these data and index file sizes:

```
-rw-rw----  1 mysql  mysql  9347072 Aug 19 11:47 person.MYD
-rw-rw----  1 mysql  mysql  8705024 Aug 19 11:47 person.MYI
```

Example of myisamchk -dvv output:

```
MyISAM file:         person
Record format:       Packed
Character set:       latin1_swedish_ci (8)
File-version:        1
Creation time:       2009-08-19 11:45:39
Recover time:        2009-08-19 11:45:50
Status:              checked,analyzed,optimized keys
Auto increment key:              1  Last value:                306688
Data records:               306688  Deleted blocks:                 0
Datafile parts:             306688  Deleted data:                   0
Datafile pointer (bytes):        4  Keyfile pointer (bytes):        3
Datafile length:           9347072  Keyfile length:           8705024
Max datafile length:    4294967294  Max keyfile length:   17179868159
Recordlength:                   52

table description:
Key Start Len Index   Type                 Rec/key         Root  Blocksize
1   2     4   unique  long                       1        99328       1024
2   6     20  multip. char packed stripped     512      6202368       1024
    26    20          char stripped            512
3   46    3   multip. uint24 NULL           306688      8704000       1024

Field Start Length Nullpos Nullbit Type
1     1     1
2     2     4                      no zeros
3     6     20                     no endspace
4     26    20                     no endspace
5     46    3      1       1       no zeros
6     49    3      1       2       no zeros
```

Explanations for the types of information myisamchk produces are given here. “Keyfile” refers to the index file. “Record” and “row” are synonymous, as are “field” and “column.”

The initial part of the table description contains these values:

- `MyISAM file`

  Name of the `MyISAM` (index) file.

- `Record format`

  The format used to store table rows. The preceding examples use `Fixed length`. Other possible values are `Compressed` and `Packed`. (`Packed` corresponds to what `SHOW TABLE STATUS` reports as `Dynamic`.)

- `Chararacter set`

  The table default character set.

- `File-version`

  Version of `MyISAM` format. Currently always 1.

- `Creation time`

  When the data file was created.

- `Recover time`

  When the index/data file was last reconstructed.

- `Status`

  Table status flags. Possible values are `crashed`, `open`, `changed`, `analyzed`, `optimized keys`, and `sorted index pages`.

- `Auto increment key`, `Last value`

  The key number associated the table's `AUTO_INCREMENT` column, and the most recently generated value for this column. These fields do not appear if there is no such column.

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 6.6.4, “`MyISAM` Table Optimization”.

- `Datafile parts`

  For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as `Data records`.

- `Deleted data`

  How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See Section 6.6.4, “`MyISAM` Table Optimization”.

- `Datafile pointer`

  The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- `Keyfile pointer`

  The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- `Max datafile length`

  How long the table data file can become, in bytes.

- `Max keyfile length`

  How long the table index file can become, in bytes.

- `Recordlength`

  How much space each row takes, in bytes.

The `table description` part of the output includes a list of all keys in the table. For each key, `myisamchk` displays some low-level information:

- `Key`

  This key's number. This value is shown only for the first column of the key. If this value is missing, the line corresponds to the second or later column of a multiple-column key. For the table shown in the example, there are two `table description` lines for the second index. This indicates that it is a multiple-part index with two parts.

- `Start`

  Where in the row this portion of the index starts.

- `Len`

  How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column. The total length of a multiple-part key is the sum of the `Len` values for all key parts.

- `Index`

  Whether a key value can exist multiple times in the index. Possible values are `unique` or `multip.` (multiple).

- `Type`

  What data type this portion of the index has. This is a `MyISAM` data type with the possible values `packed`, `stripped`, or `empty`.

- `Root`

  Address of the root index block.

- `Blocksize`

  The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- `Rec/key`

This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `myisamchk -a`. If this is not updated at all, a default value of 30 is given.

The last part of the output provides information about each column:

- `Field`

  The column number.

- `Start`

  The byte position of the column within table rows.

- `Length`

  The length of the column in bytes.

- `Nullpos`, `Nullbit`

  For columns that can be `NULL`, `MyISAM` stores `NULL` values as a flag in a byte. Depending on how many nullable columns there are, there can be one or more bytes used for this purpose. The `Nullpos` and `Nullbit` values, if nonempty, indicate which byte and bit contains that flag indicating whether the column is `NULL`.

  The position and number of bytes used to store `NULL` flags is shown in the line for field 1. This is why there are six `Field` lines for the `person` table even though it has only five columns.

- `Type`

  The data type. The value may contain any of the following descriptors:

  - `constant`

    All rows have the same value.

  - `no endspace`

    Do not store endspace.

  - `no endspace, not_always`

    Do not store endspace and do not do endspace compression for all values.

  - `no endspace, no empty`

    Do not store endspace. Do not store empty values.

  - `table-lookup`

    The column was converted to an `ENUM`.

  - `zerofill(N)`

    The most significant *`N`* bytes in the value are always 0 and are not stored.

  - `no zeros`

Do not store zeros.

- `always zero`

  Zero values are stored using one bit.

- `Huff tree`

  The number of the Huffman tree associated with the column.

- `Bits`

  The number of bits used in the Huffman tree.

The `Huff tree` and `Bits` fields are displayed if the table has been compressed with `myisampack`. See Section 4.6.4, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”, for an example of this information.

Example of `myisamchk -eiv` output:

```
Checking MyISAM file: person
Data records:  306688   Deleted blocks:       0
- check file-size
- check record delete-chain
No recordlinks
- check key delete-chain
block_size 1024:
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  98%  Packed:    0%  Max levels:  3
- check data record references index: 2
Key:  2:  Keyblocks used:  98%  Packed:   73%  Max levels:  4
- check data record references index: 3
Key:  3:  Keyblocks used:  98%  Packed:  -14%  Max levels:  3
Total:    Keyblocks used:  98%  Packed:   52%

- check records and index references
*** LOTS OF ROW NUMBERS DELETED ***

Records:            306688  M.recordlength:       25  Packed:            42%
Recordspace used:       97% Empty space:           2% Blocks/Record:   1.00
Record blocks:      306688  Delete blocks:         0
Record data:       7934464  Deleted data:          0
Lost space:         256512  Linkdata:        1156096

User time 16.39, System time 1.65
Maximum resident set size 0, Integral resident set size 0
Non-physical pagefaults 0, Physical pagefaults 0, Swaps 0
Blocks in 0 out 5, Messages in 0 out 0, Signals 0
Voluntary context switches 464, Involuntary context switches 0
Maximum memory usage: 804149 bytes (786k)
```

`myisamchk -eiv` output includes the following information:

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 6.6.4, “`MyISAM` Table Optimization”.

- `Key`

  The key number.

- `Keyblocks used`

  What percentage of the keyblocks are used. When a table has just been reorganized with `myisamchk`, the values are very high (very near theoretical maximum).

- `Packed`

  MySQL tries to pack key values that have a common suffix. This can only be used for indexes on `CHAR` and `VARCHAR` columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the preceding example, the second key is 40 bytes long and a 73% reduction in space is achieved.

- `Max levels`

  How deep the B-tree for this key is. Large tables with long key values get high values.

- `Records`

  How many rows are in the table.

- `M.recordlength`

  The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- `Packed`

  MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage of savings achieved by doing this.

- `Recordspace used`

  What percentage of the data file is used.

- `Empty space`

  What percentage of the data file is unused.

- `Blocks/Record`

  Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See Section 6.6.4, “`MyISAM` Table Optimization”.

- `Recordblocks`

  How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- `Deleteblocks`

  How many blocks (links) are deleted.

- `Recorddata`

  How many bytes in the data file are used.

- `Deleted data`

  How many bytes in the data file are deleted (unused).

- `Lost space`

  If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- `Linkdata`

  When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

4.6.2.6 `myisamchk` Memory Usage

Memory allocation is important when you run `myisamchk`. `myisamchk` uses no more memory than its memory-related variables are set to. If you are going to use `myisamchk` on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get `myisamchk` to operate faster. For example, if you have more than 512MB RAM available, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --sort_buffer_size=256M \
           --key_buffer_size=512M \
           --read_buffer_size=64M \
           --write_buffer_size=64M ...
```

Using `--sort_buffer_size=16M` is probably enough for most cases.

Be aware that `myisamchk` uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory file system, out of memory errors can easily occur. If this happens, run `myisamchk` with the `--tmpdir=path` [318] option to specify a directory located on a file system that has more space.

When performing repair operations, `myisamchk` also needs a lot of disk space:

- Twice the size of the data file (the original file and a copy). This space is not needed if you do a repair with `--quick` [317]; in this case, only the index file is re-created. *This space must be available on the same file system as the original data file*, as the copy is created in the same directory as the original.

- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space must be available on the same file system as the original data file.

- When using `--recover` [317] or `--sort-recover` [318] (but not when using `--safe-recover` [318]), you need space on disk for sorting. This space is allocated in the temporary directory (specified by `TMPDIR` or `--tmpdir=path` [318]). The following formula yields the amount of space required:

  ```
  (largest_key + row_pointer_length) * number_of_rows * 2
  ```

  You can check the length of the keys and the *`row_pointer_length`* with `myisamchk -dv tbl_name` (see Section 4.6.2.5, “Obtaining Table Information with `myisamchk`”). The *`row_pointer_length`* and *`number_of_rows`* values are the `Datafile pointer` and `Data records` values in the table description. To determine the *`largest_key`* value, check the `Key` lines in the table description. The `Len` column indicates the number of bytes for each key part. For a multiple-column index, the key size is the sum of the `Len` values for all key parts.

If you have a problem with disk space during repair, you can try `--safe-recover` [318] instead of `--recover` [317].

4.6.3 myisamlog — Display MyISAM Log File Contents

`myisamlog` processes the contents of a `MyISAM` log file. `isamlog` is similar, but is used with `ISAM` log files.

Invoke `myisamlog` or `isamlog`like this:

```
shell> myisamlog [options] [log_file [tbl_name] ...]
shell> isamlog [options] [log_file [tbl_name] ...]
```

The default operation is update (`-u`). If a recovery is done (`-r`), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is `myisam.log` for `myisamlog` and `isam.log` for `isamlog` if no *`log_file`* argument is given. If tables are named on the command line, only those tables are updated.

`myisamlog` and `isamlog` understand the following options:

- `-?`, `-I`

  Display a help message and exit.

- `-c N`

  Execute only *`N`* commands.

- `-f N`

  Specify the maximum number of open files.

- `-i`

  Display extra information before exiting.

- `-o offset`

  Specify the starting offset.

- `-p N`

  Remove *`N`* components from path.

- `-r`

  Perform a recovery operation.

- `-R record_pos_file record_pos`

  Specify record position file and record position.

- `-u`

  Perform an update operation.

- `-v`

Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.

- `-w write_file`

  Specify the write file.

- `-V`

  Display version information.

4.6.4 myisampack — Generate Compressed, Read-Only MyISAM Tables

The `myisampack` utility compresses `MyISAM` tables. `myisampack` works by compressing each column in the table separately. Usually, `myisampack` packs the data file 40% to 70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

A similar utility, `pack_isam`, compresses `ISAM` tables. Because `ISAM` tables are deprecated, this section discusses only `myisampack`, but the general procedures for using `myisampack` are also true for `pack_isam` unless otherwise specified. References to `myisamchk` should be read as references to `isamchk` if you are using `pack_isam`.

Please note the following:

- If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.
- After packing a table, it becomes read only. This is generally intended (such as when accessing packed tables on a CD).
- `myisampack` can pack `BLOB` or `TEXT` columns. (The older `pack_isam` program for `ISAM` tables does not have this capability.)
- `myisampack` does not support partitioned tables.

Invoke `myisampack` like this:

```
shell> myisampack [options] file_name ...
```

Each file name argument should be the name of an index (`.MYI`) file. If you are not in the database directory, you should specify the path name to the file. It is permissible to omit the `.MYI` extension.

After you compress a table with `myisampack`, you should use `myisamchk -rq` to rebuild its indexes. Section 4.6.2, “myisamchk — MyISAM Table-Maintenance Utility”.

`myisampack` supports the following options. It also reads option files and supports the options for processing them described at Command-Line Options that Affect Option-File Handling.

- `--help` [328], `-?`

  Display a help message and exit.

- `--backup` [329], `-b`

  Make a backup of each table's data file using the name *`tbl_name`*`.OLD`.

- `--character-sets-dir=`*`path`* [329]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--debug[=`*`debug_options`*`]` [329], `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `'d:t:o,`*`file_name`*`'`. The default is `'d:t:o'`.

- `--force` [329], `-f`

  Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of `myisampack` exists. (`myisampack` creates an intermediate file named *`tbl_name`*`.TMD` in the database directory while it compresses the table. If you kill `myisampack`, the `.TMD` file might not be deleted.) Normally, `myisampack` exits with an error if it finds that *`tbl_name`*`.TMD` exists. With `--force` [329], `myisampack` packs the table anyway.

- `--join=`*`big_tbl_name`* [329], `-j` *`big_tbl_name`*

  Join all tables named on the command line into a single packed table *`big_tbl_name`*. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

  `big_tbl_name` must not exist prior to the join operation. All source tables named on the command line to be merged into *`big_tbl_name`* must exist. The source tables are read for the join operation but not modified. The join operation does not create a `.frm` file for *`big_tbl_name`*, so after the join operation finishes, copy the `.frm` file from one of the source tables and name it *`big_tbl_name`*`.frm`.

- `--packlength=`*`len`*, `-p` *`len`*

  (`pack_isam` only) Specify the row length storage size, in bytes. The value should be 1, 2, or 3. `pack_isam` stores all rows with length pointers of 1, 2, or 3 bytes. In most normal cases, `pack_isam` can determine the correct length value before it begins packing the file, but it may notice during the packing process that it could have used a shorter length. In this case, `pack_isam` prints a note that you could use a shorter row length the next time you pack the same file.

- `--silent` [329], `-s`

  Silent mode. Write output only when errors occur.

- `--test` [329], `-t`

  Do not actually pack the table, just test packing it.

- `--tmpdir=`*`path`* [329], `-T` *`path`*

  Use the named directory as the location where `myisampack` creates temporary files.

- `--verbose` [329], `-v`

  Verbose mode. Write information about the progress of the packing operation and its result.

- `--version` [329], `-V`

  Display version information and exit.

- `--wait` [330], `-w`

  Wait and retry if the table is in use. If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r--   1 monty    my         994128 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          53248 Apr 17 19:00 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-02-02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile parts:            1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long        1024       1024          1
2   32    30  multip. text                10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
7     32    30
8     62    35
9     97    35
10    132   35
11    167   4
12    171   16
13    187   35
14    222   4
15    226   16
16    242   20
17    262   20
18    282   20
19    302   30
20    332   4
21    336   4
22    340   1
23    341   8
24    349   8
25    357   8
26    365   2
27    367   2
28    369   4
29    373   4
30    377   1
31    378   2
32    380   8
```

```
33    388   4
34    392   4
35    396   4
36    400   4
37    404   1
38    405   4
39    409   4
40    413   4
41    417   4
42    421   4
43    425   4
44    429   20
45    449   30
46    479   1
47    480   1
48    481   79
49    560   79
50    639   79
51    718   79
52    797   8
53    805   1
54    806   1
55    807   20
56    827   4
57    831   4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:     20  empty-space:   16  empty-zero:     12  empty-fill:  11
pre-space:   0  end-space:     12  table-lookups:   5  zero:         7
Original trees:  57  After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> ls -l station.*
-rw-rw-r--   1 monty    my         127874 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          55296 Apr 17 19:04 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-04-17 19:04:26
Data records:               1192  Deleted blocks:              0
Datafile parts:             1192  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:    16777215  Max keyfile length:     131071
Recordlength:                834
Record format: Compressed

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long       10240       1024          1
2   32    30  multip. text                54272       1024          1

Field Start Length Type                          Huff tree  Bits
1     1     1      constant                            1     0
2     2     4      zerofill(1)                         2     9
3     6     4      no zeros, zerofill(1)               2     9
4     10    1                                          3     9
5     11    20     table-lookup                        4     0
6     31    1                                          3     9
```

```
7     32    30     no endspace, not_always              5     9
8     62    35     no endspace, not_always, no empty    6     9
9     97    35     no empty                             7     9
10    132   35     no endspace, not_always, no empty    6     9
11    167   4      zerofill(1)                          2     9
12    171   16     no endspace, not_always, no empty    5     9
13    187   35     no endspace, not_always, no empty    6     9
14    222   4      zerofill(1)                          2     9
15    226   16     no endspace, not_always, no empty    5     9
16    242   20     no endspace, not_always              8     9
17    262   20     no endspace, no empty                8     9
18    282   20     no endspace, no empty                5     9
19    302   30     no endspace, no empty                6     9
20    332   4      always zero                          2     9
21    336   4      always zero                          2     9
22    340   1                                           3     9
23    341   8      table-lookup                         9     0
24    349   8      table-lookup                        10     0
25    357   8      always zero                          2     9
26    365   2                                           2     9
27    367   2      no zeros, zerofill(1)                2     9
28    369   4      no zeros, zerofill(1)                2     9
29    373   4      table-lookup                        11     0
30    377   1                                           3     9
31    378   2      no zeros, zerofill(1)                2     9
32    380   8      no zeros                             2     9
33    388   4      always zero                          2     9
34    392   4      table-lookup                        12     0
35    396   4      no zeros, zerofill(1)               13     9
36    400   4      no zeros, zerofill(1)                2     9
37    404   1                                           2     9
38    405   4      no zeros                             2     9
39    409   4      always zero                          2     9
40    413   4      no zeros                             2     9
41    417   4      always zero                          2     9
42    421   4      no zeros                             2     9
43    425   4      always zero                          2     9
44    429   20     no empty                             3     9
45    449   30     no empty                             3     9
46    479   1                                          14     4
47    480   1                                          14     4
48    481   79     no endspace, no empty               15     9
49    560   79     no empty                             2     9
50    639   79     no empty                             2     9
51    718   79     no endspace                         16     9
52    797   8      no empty                             2     9
53    805   1                                          17     1
54    806   1                                           3     9
55    807   20     no empty                             3     9
56    827   4      no zeros, zerofill(2)                2     9
57    831   4      no zeros, zerofill(1)                2     9
```

`myisampack` displays the following kinds of information:

- `normal`

  The number of columns for which no extra packing is used.

- `empty-space`

  The number of columns containing values that are only spaces. These occupy one bit.

- `empty-zero`

  The number of columns containing values that are only binary zeros. These occupy one bit.

- `empty-fill`

  The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.

- `pre-space`

  The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- `end-space`

  The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- `table-lookup`

  The column had only a small number of different values, which were converted to an `ENUM` before Huffman compression.

- `zero`

  The number of columns for which all values are zero.

- `Original trees`

  The initial number of Huffman trees.

- `After join`

  The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, the `Field` lines displayed by `myisamchk -dvv` include additional information about each column:

- `Type`

  The data type. The value may contain any of the following descriptors:

  - `constant`

    All rows have the same value.

  - `no endspace`

    Do not store endspace.

  - `no endspace, not_always`

    Do not store endspace and do not do endspace compression for all values.

  - `no endspace, no empty`

    Do not store endspace. Do not store empty values.

  - `table-lookup`

    The column was converted to an `ENUM`.

- `zerofill(N)`

  The most significant *N* bytes in the value are always 0 and are not stored.

- `no zeros`

  Do not store zeros.

- `always zero`

  Zero values are stored using one bit.

- `Huff tree`

  The number of the Huffman tree associated with the column.

- `Bits`

  The number of bits used in the Huffman tree.

After you run `myisampack`, you must run `myisamchk` to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

A similar procedure applies for `ISAM` tables. After using `pack_isam`, use `isamchk` to re-create the indexes:

```
shell> isamchk -rq --sort-index --analyze tbl_name.ISM
```

After you have installed the packed table into the MySQL database directory, you should execute `mysqladmin flush-tables` to force `mysqld` to start using the new table.

To unpack a packed table, use the `--unpack` [318] option to `myisamchk` or `isamchk`.

4.6.5 `mysqlaccess` — Client for Checking Access Privileges

`mysqlaccess` is a diagnostic tool that Yves Carlier has provided for the MySQL distribution. It checks the access privileges for a host name, user name, and database combination. Note that `mysqlaccess` checks access using only the `user`, `db`, and `host` tables. It does not check table, column, or routine privileges specified in the `tables_priv` or `columns_priv` tables.

Invoke `mysqlaccess` like this:

```
shell> mysqlaccess [host_name [user_name [db_name]]] [options]
```

`mysqlaccess` supports the following options.

Table 4.9 `mysqlaccess` Options

Format	Option File	Description
--brief [335]	brief [335]	Generate reports in single-line tabular format
--commit [335]	commit [335]	Copy the new access privileges from the temporary tables to the original grant tables

Format	Option File	Description
--copy [335]	copy [335]	Reload the temporary grant tables from original ones
--db=db_name [335]	db [335]	Specify the database name
--debug=# [336]	debug [336]	Specify the debug level
--help [335]		Display help message and exit
--host=host_name [336]	host [336]	Connect to the MySQL server on the given host
--howto [336]	howto [336]	Display some examples that show how to use mysqlaccess
--old_server [336]	old_server [336]	Assume that the server is an old MySQL server (prior to MySQL 3.21)
--password[=password] [336]	password [336]	The password to use when connecting to the server
--plan [336]	plan [336]	Display suggestions and ideas for future releases
--preview [336]	preview [336]	Show the privilege differences after making changes to the temporary grant tables
--relnotes [336]	relnotes [336]	Display the release notes
--rhost=host_name [336]	rhost [336]	Connect to the MySQL server on the given host
--rollback [336]	rollback [336]	Undo the most recent changes to the temporary grant tables.
--spassword[=password] [336]	spassword [336]	The password to use when connecting to the server as the superuser
--superuser=user_name [336]	superuser [336]	Specify the user name for connecting as the superuser
--table [337]	table [337]	Generate reports in table format
--user=user_name, [337]	user [337]	MySQL user name to use when connecting to server
--version [337]		Display version information and exit

- `--help` [335], `-?`

  Display a help message and exit.

- `--brief` [335], `-b`

  Generate reports in single-line tabular format.

- `--commit` [335]

  Copy the new access privileges from the temporary tables to the original grant tables. The grant tables must be flushed for the new privileges to take effect. (For example, execute a `mysqladmin reload` command.)

- `--copy` [335]

  Reload the temporary grant tables from original ones.

- `--db=`*`db_name`* [335], `-d` *`db_name`*

Specify the database name.

- `--debug=N` [336]

  Specify the debug level. *`N`* can be an integer from 0 to 3.

- `--host=host_name` [336], `-h host_name`

  The host name to use in the access privileges.

- `--howto` [336]

  Display some examples that show how to use `mysqlaccess`.

- `--old_server` [336]

  Assume that the server is an old MySQL server (before MySQL 3.21) that does not yet know how to handle full `WHERE` clauses.

- `--password[=password]` [336], `-p[password]`

  The password to use when connecting to the server. If you omit the *`password`* value following the `--password` [336] or `-p` option on the command line, `mysqlaccess` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”.

- `--plan` [336]

  Display suggestions and ideas for future releases.

- `--preview` [336]

  Show the privilege differences after making changes to the temporary grant tables.

- `--relnotes` [336]

  Display the release notes.

- `--rhost=host_name` [336], `-H host_name`

  Connect to the MySQL server on the given host.

- `--rollback` [336]

  Undo the most recent changes to the temporary grant tables.

- `--spassword[=password]` [336], `-P[password]`

  The password to use when connecting to the server as the superuser. If you omit the *`password`* value following the `--spassword` [336] or `-p` option on the command line, `mysqlaccess` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”.

- `--superuser=user_name` [336], `-U user_name`

  Specify the user name for connecting as the superuser.

- `--table` [337], `-t`

  Generate reports in table format.

- `--user=user_name` [337], `-u user_name`

  The user name to use in the access privileges.

- `--version` [337], `-v`

  Display version information and exit.

If your MySQL distribution is installed in some nonstandard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error will occur when you run `mysqlaccess`.

4.6.6 mysqlbinlog — Utility for Processing Binary Log Files

The server's binary log consists of files containing “events” that describe modifications to database contents. The server writes these files in binary format. To display their contents in text format, use the `mysqlbinlog` utility, which is available as of MySQL 3.23.14. You can also use `mysqlbinlog` to display the contents of relay log files written by a slave server in a replication setup because relay logs have the same format as binary logs. The binary log and relay log are discussed further in Section 5.3.4, “The Binary Log”, and Section 14.3.1, “Replication Relay and Status Files”.

Invoke `mysqlbinlog` like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named `binlog.000003`, use this command:

```
shell> mysqlbinlog binlog.0000003
```

The output includes events contained in `binlog.000003`. Event information includes the SQL statement, the ID of the server on which it was executed, the timestamp when the statement was executed, how much time it took, and so forth.

Events are preceded by header comments that provide additional information. For example:

```
# at 141
#100309  9:28:36 server id 123  end_log_pos 245
  Query thread_id=3350  exec_time=11  error_code=0
```

In the first line, the number following `at` indicates the starting position of the event in the binary log file.

The second line starts with a date and time indicating when the statement started on the server where the event originated. For replication, this timestamp is propagated to slave servers. `server id` is the `server_id` [426] value of the server where the event originated. `end_log_pos` indicates where the next event starts (that is, it is the end position of the current event + 1). `thread_id` indicates which thread executed the event. `exec_time` is the time spent executing the event, on a master server. On a slave, it is the difference of the end execution time on the slave minus the beginning execution time on the master.

The difference serves as an indicator of how much replication lags behind the master. error_code indicates the result from executing the event. Zero means that no error occurred.

The output from mysqlbinlog can be re-executed (for example, by using it as input to mysql) to redo the statements in the log. This is useful for recovery operations after a server crash. For other usage examples, see the discussion later in this section and in Section 6.5, "Point-in-Time (Incremental) Recovery Using the Binary Log".

Normally, you use mysqlbinlog to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the --read-from-remote-server [341] option. To read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are --host [340], --password [341], --port [341], --protocol [341], --socket [341], and --user [342]; they are ignored except when you also use the --read-from-remote-server [341] option.

mysqlbinlog supports the following options, which can be specified on the command line or in the [mysqlbinlog] and [client] option file groups. mysqlbinlog also supports the options for processing option files described at Command-Line Options that Affect Option-File Handling.

Table 4.10 mysqlbinlog Options

Format	Option File	Description	Introduced	Deprecated
--character-sets-dir=path [339]	character-sets-dir [339]	The directory where character sets are installed		
--database=db_name [339]	database [339]	List entries for just this database		
--debug[=debug_options] [340]	debug [340]	Write a debugging log		
--disable-log-bin [340]	disable-log-bin [340]	Disable binary logging		
--force-read [340]	force-read [340]	If mysqlbinlog reads a binary log event that it does not recognize, it prints a warning		
--help [339]		Display help message and exit		
--host=host_name [340]	host [340]	Connect to the MySQL server on the given host		
--local-load=path [340]	local-load [340]	Prepare local temporary files for LOAD DATA INFILE in the specified directory		
--offset=# [340]	offset [340]	Skip the first N entries in the log		
--password[=password] [341]	password [341]	The password to use when connecting to the server		
--port=port_num [341]	port [341]	The TCP/IP port number to use for the connection		
--position=# [341]	position [341]	Deprecated. Use --start-position		4.1.3
--protocol=type [341]	protocol [341]	The connection protocol to use		
--read-from-remote-server [341]	read-from-remote-server [341]	Read binary log from MySQL server rather than local log file		

Format	Option File	Description	Introduced	Deprecated
--result-file=name [341]	result-file [341]	Direct output to the given file		
--set-charset=charset_name [341]	set-charset [341]	Add a SET NAMES charset_name statement to the output	4.1.21	
--short-form [341]	short-form [341]	Display only the statements contained in the log		
--socket=path [341]	socket [341]	For connections to localhost		
--start-datetime=datetime [341]	start-datetime [341]	Read binary log from first event with timestamp equal to or later than datetime argument		
--start-position=# [342]	start-position [342]	Read binary log from first event with position equal to or greater than argument	4.1.4	
--stop-datetime=datetime [342]	stop-datetime [342]	Stop reading binary log at first event with timestamp equal to or greater than datetime argument	4.1.4	
--stop-position=# [342]	stop-position [342]	Stop reading binary log at first event with position equal to or greater than argument	4.1.4	
--to-last-log [342]	to-last-log [342]	Do not stop at the end of requested binary log from a MySQL server, but rather continue printing to end of last binary log	4.1.2	
--user=user_name, [342]	user [342]	MySQL user name to use when connecting to server		
--version [342]		Display version information and exit		

- `--help` [339], `-?`

  Display a help message and exit.

- `--character-sets-dir=`*`path`* [339]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--database=`*`db_name`* [339], `-d` *`db_name`*

  This option causes `mysqlbinlog` to output entries from the binary log (local log only) that occur while *`db_name`* is been selected as the default database by `USE`.

  The `--database` [339] option for `mysqlbinlog` is similar to the `--binlog-do-db` [1182] option for `mysqld`, but can be used to specify only one database. If `--database` [339] is given multiple times, only the last instance is used.

  The `--database` [339] option works as follows:

  - While *`db_name`* is the default database, statements are output whether they modify tables in *`db_name`* or a different database.
  - Unless *`db_name`* is selected as the default database, statements are not output, even if they modify tables in *`db_name`*.

- There is an exception for CREATE DATABASE, ALTER DATABASE, and DROP DATABASE. The database being *created, altered, or dropped* is considered to be the default database when determining whether to output the statement.

Suppose that the binary log contains these statements:

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j)  VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i)      VALUES(102);
INSERT INTO db2.t2 (j)  VALUES(201);
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j)  VALUES(202);
INSERT INTO t2 (j)      VALUES(203);
```

mysqlbinlog --database=test does not output the first two INSERT statements because there is no default database. It outputs the three INSERT statements following USE test, but not the three INSERT statements following USE db2.

mysqlbinlog --database=db2 does not output the first two INSERT statements because there is no default database. It does not output the three INSERT statements following USE test, but does output the three INSERT statements following USE db2.

- --debug[=*debug_options*] [340], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is 'd:t:o,*file_name*'. The default is 'd:t:o,/tmp/mysqlbinlog.trace'.

- --disable-log-bin [340], -D

  Disable binary logging. This is useful for avoiding an endless loop if you use the --to-last-log [342] option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged. This option is available as of MySQL 4.1.8.

  This option requires that you have the SUPER [493] privilege. It causes mysqlbinlog to include a SET sql_log_bin = 0 statement in its output to disable binary logging of the remaining output. The SET statement is ineffective unless you have the SUPER [493] privilege.

- --force-read [340], -f

  With this option, if mysqlbinlog reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, mysqlbinlog stops if it reads such an event.

- --host=*host_name* [340], -h *host_name*

  Get the binary log from the MySQL server on the given host.

- --local-load=*path* [340], -l *path*

  Prepare local temporary files for LOAD DATA INFILE in the specified directory.

- --offset=*N* [340], -o *N*

  Skip the first *N* entries in the log.

- `--password[=password]` [341], `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` [341] or `-p` option on the command line, `mysqlbinlog` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, "End-User Guidelines for Password Security". You can use an option file to avoid giving the password on the command line.

- `--port=port_num` [341], `-P port_num`

  The TCP/IP port number to use for connecting to a remote server.

- `--position=N` [341]

  Deprecated. Use `--start-position` [342] instead (starting from MySQL 4.1.4).

- `--protocol={TCP|SOCKET|PIPE|MEMORY}` [341]

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, "Connecting to the MySQL Server". This option was added in MySQL 4.1.

- `--read-from-remote-server` [341], `-R`

  Read the binary log from a MySQL server rather than reading a local log file. Any connection parameter options are ignored unless this option is given as well. These options are `--host` [340], `--password` [341], `--port` [341], `--protocol` [341], `--socket` [341], and `--user` [342].

  This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files.

- `--result-file=name` [341], `-r name`

  Direct output to the given file.

- `--set-charset=charset_name` [341]

  Add a `SET NAMES charset_name` statement to the output to specify the character set to be used for processing log files. This option was added in MySQL 4.1.21.

- `--short-form` [341], `-s`

  Display only the statements contained in the log, without any extra information. This is for testing only, and should not be used in production systems.

- `--socket=path` [341], `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--start-datetime=datetime` [341]

  Start reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. The *`datetime`* value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

```
shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
```

This option is available as of MySQL 4.1.4. It is useful for point-in-time recovery. See Section 6.3, “Example Backup and Recovery Strategy”.

- `--start-position=N` [342], `-j N`

  Start reading the binary log at the first event having a position equal to or greater than *`N`*. This option applies to the first log file named on the command line. Available as of MySQL 4.1.4 (previously named `--position` [341]).

  This option is useful for point-in-time recovery. See Section 6.3, “Example Backup and Recovery Strategy”.

- `--stop-datetime=datetime` [342]

  Stop reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` [341] option for information about the *`datetime`* value. This option is available as of MySQL 4.1.4.

  This option is useful for point-in-time recovery. See Section 6.3, “Example Backup and Recovery Strategy”.

- `--stop-position=N` [342]

  Stop reading the binary log at the first event having a position equal to or greater than *`N`*. This option applies to the last log file named on the command line. Available as of MySQL 4.1.4.

  This option is useful for point-in-time recovery. See Section 6.3, “Example Backup and Recovery Strategy”.

- `--to-last-log` [342], `-t`

  Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server` [341]. Available as of MySQL 4.1.2.

- `--user=user_name` [342], `-u user_name`

  The MySQL user name to use when connecting to a remote server.

- `--version` [342], `-V`

  Display version information and exit.

You can also set the following variable by using `--var_name=value` syntax:

- `open_files_limit`

  Specify the number of open file descriptors to reserve.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated*.

You can pipe the output of mysqlbinlog into the mysql client to execute the events contained in the binary log. This technique is used to recover from a crash when you have an old backup (see Section 6.5, "Point-in-Time (Incremental) Recovery Using the Binary Log"). For example:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

You can also redirect the output of mysqlbinlog to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the mysql program:

```
shell> mysqlbinlog binlog.000001 > tmpfile
shell> ... edit tmpfile ...
shell> mysql -u root -p < tmpfile
```

When mysqlbinlog is invoked with the --start-position [342] option, it displays only those events with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the --stop-datetime [342] option (to be able to say, for example, "roll forward my databases to how they were today at 10:30 a.m.").

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using multiple connections to the server causes problems if the first log file contains a CREATE TEMPORARY TABLE statement and the second log contains a statement that uses the temporary table. When the first mysql process terminates, the server drops the temporary table. When the second mysql process attempts to use the table, the server reports "unknown table."

To avoid problems like this, use a *single* mysql process to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

In MySQL 3.23, the binary log did not contain the data to load for LOAD DATA INFILE statements. To execute such a statement from a binary log file, the original data file was needed. Starting from MySQL 4.0.14, the binary log does contain the data, so mysqlbinlog can produce output that reproduces the LOAD DATA INFILE operation without the original data file. mysqlbinlog copies the data to a temporary file and writes a LOAD DATA LOCAL INFILE statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the --local-load [340] option.

Because mysqlbinlog converts LOAD DATA INFILE statements to LOAD DATA LOCAL INFILE statements (that is, it adds LOCAL), both the client and the server that you use to process the statements must be configured with the LOCAL capability enabled. See Section 5.4.5, “Security Issues with LOAD DATA LOCAL”.

Warning

The temporary files created for LOAD DATA LOCAL statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like *original_file_name-#-#*.

Before MySQL 4.1, mysqlbinlog could not prepare output suitable for mysql if the binary log contained interlaced statements originating from different clients that used temporary tables of the same name. This is fixed in MySQL 4.1. However, the problem still existed for LOAD DATA INFILE statements until it was fixed in MySQL 4.1.8.

4.6.7 mysqldumpslow — Summarize Slow Query Log Files

The MySQL slow query log contains information about queries that take a long time to execute (see Section 5.3.5, “The Slow Query Log”). mysqldumpslow parses MySQL slow query log files and prints a summary of their contents.

Normally, mysqldumpslow groups queries that are similar except for the particular values of number and string data values. It “abstracts” these values to N and 'S' when displaying summary output. The -a and -n options can be used to modify value abstracting behavior.

Invoke mysqldumpslow like this:

```
shell> mysqldumpslow [options] [log_file ...]
```

mysqldumpslow supports the following options.

Table 4.11 mysqldumpslow Options

Format	Option File	Description
-a [345]		Do not abstract all numbers to N and strings to S
-n num [345]		Abstract numbers with at least the specified digits
--debug [345]	debug [345]	Write debugging information
-g pattern [345]		Only consider statements that match the pattern
--help [344]		Display help message and exit
-h name [345]		Host name of the server in the log file name
-i name [345]		Name of the server instance
-l [345]		Do not subtract lock time from total time
-r [345]		Reverse the sort order
-s value [345]		How to sort output
-t num [345]		Display only first num queries
--verbose [345]	verbose [345]	Verbose mode

- --help [344]

Display a help message and exit.

- `-a`

  Do not abstract all numbers to `N` and strings to `'S'`.

- `--debug` [345], `-d`

  Run in debug mode.

- `-g` *`pattern`*

  Consider only queries that match the (`grep`-style) pattern.

- `-h` *`host_name`*

  Host name of MySQL server for `*-slow.log` file name. The value can contain a wildcard. The default is `*` (match all).

- `-i` *`name`*

  Name of server instance (if using `mysql.server` startup script).

- `-l`

  Do not subtract lock time from total time.

- `-n` *`N`*

  Abstract numbers with at least *`N`* digits within names.

- `-r`

  Reverse the sort order.

- `-s` *`sort_type`*

  How to sort the output. The value of *`sort_type`* should be chosen from the following list:

  - `t`, `at`: Sort by query time or average query time
  - `l`, `al`: Sort by lock time or average lock time
  - `s`, `as`: Sort by rows sent or average rows sent
  - `c`: Sort by count

  By default, `mysqldumpslow` sorts by average query time (equivalent to `-s at`).

- `-t` *`N`*

  Display only the first *`N`* queries in the output.

- `--verbose` [345], `-v`

  Verbose mode. Print more information about what the program does.

Example of usage:

```
shell> mysqldumpslow

Reading mysql slow query log from /usr/local/mysql/data/mysqld51-apple-slow.log
Count: 1  Time=4.32s (4s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1

Count: 3  Time=2.53s (7s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1 limit N

Count: 3  Time=2.13s (6s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t1 select * from t1
```

4.6.8 mysqlhotcopy — A Database Backup Program

`mysqlhotcopy` is a Perl script that was originally written and contributed by Tim Bunce. It uses `FLUSH TABLES`, `LOCK TABLES`, and `cp` or `scp` to make a database backup. It is a fast way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. `mysqlhotcopy` works only for backing up `MyISAM` and `ISAM` tables, and `ARCHIVE` tables as of MySQL 4.1. `mysqlhotcopy` runs on Unix, and also on NetWare as of MySQL 4.0.18.

To use `mysqlhotcopy`, you must have read access to the files for the tables that you are backing up, the `SELECT` [492] privilege for those tables, the `RELOAD` [492] privilege (to be able to execute `FLUSH TABLES`), and the `LOCK TABLES` [492] privilege (to be able to lock the tables).

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde (“~”):

```
shell> mysqlhotcopy db_name./~regex/
```

`mysqlhotcopy` supports the following options, which can be specified on the command line or in the `[mysqlhotcopy]` and `[client]` option file groups.

Table 4.12 mysqlhotcopy Options

Format	Option File	Description
--addtodest [347]	addtodest [347]	Do not rename target directory (if it exists); merely add files to it
--allowold [347]	allowold [347]	Do not abort if a target exists; rename it by adding an _old suffix
--checkpoint=db_name.tbl_name [347]	checkpoint [347]	Insert checkpoint entries
--chroot=path [347]	chroot [347]	Base directory of the chroot jail in which mysqld operates
--debug [347]	debug [347]	Write a debugging log
--dryrun [347]	dryrun [347]	Report actions without performing them

Format	Option File	Description
--flushlog [348]	flushlog [348]	Flush logs after all tables are locked
--help [347]		Display help message and exit
--host=host_name [348]	host [348]	Connect to the MySQL server on the given host
--keepold [348]	keepold [348]	Do not delete previous (renamed) target when done
--method [348]	method [348]	The method for copying files
--noindices [348]	noindices [348]	Do not include full index files in the backup
--password[=password] [348]	password [348] [348]	The password to use when connecting to the server
--port=port_num [348]	port [348]	The TCP/IP port number to use for the connection
--quiet [348]	quiet [348]	Be silent except for errors
--regexp [348]	regexp [348]	Copy all databases with names that match the given regular expression
--resetmaster [348]	resetmaster [348]	Reset the binary log after locking all the tables
--resetslave [348]	resetslave [348]	Reset the master.info file after locking all the tables
--socket=path [348]	socket [348]	For connections to localhost
--tmpdir=path [349]	tmpdir [349]	The temporary directory
--user=user_name, [349]	user [349]	MySQL user name to use when connecting to server

- `--help` [347], `-?`

  Display a help message and exit.

- `--addtodest` [347]

  Do not rename target directory (if it exists); merely add files to it. This option was added in MySQL 4.0.13.

- `--allowold` [347]

  Do not abort if a target exists; rename it by adding an `_old` suffix.

- `--checkpoint=`*`db_name.tbl_name`* [347]

  Insert checkpoint entries into the specified database *`db_name`* and table *`tbl_name`*.

- `--chroot=`*`path`* [347]

  Base directory of the `chroot` jail in which `mysqld` operates. The *`path`* value should match that of the `--chroot` [385] option given to `mysqld`. This option was added in MySQL 4.0.19.

- `--debug` [347]

  Enable debug output.

- `--dryrun` [347], `-n`

  Report actions without performing them.

- `--flushlog` [348]

  Flush logs after all tables are locked.

- `--host=host_name` [348], `-h host_name`

  The host name of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to `localhost` using a Unix socket file.

- `--keepold` [348]

  Do not delete previous (renamed) target when done.

- `--method=command` [348]

  The method for copying files (`cp` or `scp`). The default is `cp`.

- `--noindices` [348]

  Do not include full index files for `MyISAM` and `ISAM` tables in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be reconstructed later with `myisamchk -rq` for `MyISAM` tables or `isamchk -rq` for `ISAM` tables.

- `--password=password` [348], `-ppassword`

  The password to use when connecting to the server. The password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--port=port_num` [348], `-P port_num`

  The TCP/IP port number to use when connecting to the local server.

- `--quiet` [348], `-q`

  Be silent except for errors.

- `--record_log_pos=db_name.tbl_name` [348]

  Record master and slave status in the specified database *`db_name`* and table *`tbl_name`*.

- `--regexp=expr` [348]

  Copy all databases with names that match the given regular expression.

- `--resetmaster` [348]

  Reset the binary log after locking all the tables.

- `--resetslave` [348]

  Reset the `master.info` file after locking all the tables.

- `--socket=path` [348], `-S path`

  The Unix socket file to use for connections to `localhost`.

- `--suffix=str` [349]

  The suffix to use for names of copied databases.

- `--tmpdir=path` [349]

  The temporary directory. The default is `/tmp`.

- `--user=user_name` [349], `-u user_name`

  The MySQL user name to use when connecting to the server.

Use `perldoc` for additional `mysqlhotcopy` documentation, including information about the structure of the tables needed for the `--checkpoint` [347] and `--record_log_pos` [348] options:

```
shell> perldoc mysqlhotcopy
```

4.6.9 mysqlmanagerc — Internal Test-Suite Program

This program was used internally for test purposes. As of MySQL 5.0, it is no longer used.

4.6.10 mysqlmanager-pwgen — Internal Test-Suite Program

This program was used internally for test purposes. As of MySQL 5.0, it is no longer used.

4.6.11 mysql_convert_table_format — Convert Tables to Use a Given Storage Engine

`mysql_convert_table_format` converts the tables in a database to use a particular storage engine (`MyISAM` by default). `mysql_convert_table_format` is written in Perl and requires that the `DBI` and `DBD::mysql` Perl modules be installed (see Section 2.14, “Perl Installation Notes”).

Invoke `mysql_convert_table_format` like this:

```
shell> mysql_convert_table_format [options]db_name
```

The *db_name* argument indicates the database containing the tables to be converted.

`mysql_convert_table_format` supports the options described in the following list.

- `--help` [349]

  Display a help message and exit.

- `--force` [349]

  Continue even if errors occur.

- `--host=host_name` [349]

  Connect to the MySQL server on the given host.

- `--password=password` [349]

  The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--port=port_num` [350]

  The TCP/IP port number to use for the connection.

- `--socket=path` [350]

  For connections to `localhost`, the Unix socket file to use.

- `--type=engine_name` [350]

  Specify the storage engine that the tables should be converted to use. The default is `MyISAM` if this option is not given.

- `--user=user_name` [350]

  The MySQL user name to use when connecting to the server.

- `--verbose` [350]

  Verbose mode. Print more information about what the program does.

- `--version` [350]

  Display version information and exit.

4.6.12 mysql_explain_log — Use EXPLAIN on Statements in Query Log

`mysql_explain_log` reads its standard input for query log contents. It uses `EXPLAIN` to analyze `SELECT` statements found in the input. `UPDATE` statements are rewritten to `SELECT` statements and also analyzed with `EXPLAIN`. `mysql_explain_log` then displays a summary of its results.

The results may assist you in determining which queries result in table scans and where it would be beneficial to add indexes to your tables.

Invoke `mysql_explain_log` like this, where *`log_file`* contains all or part of a MySQL query log:

```
shell> mysql_explain_log [options] < log_file
```

`mysql_explain_log` supports the following options:

- `--help` [350], `-?`

  Display a help message and exit.

- `--date=YYMMDD` [350], `-d YYMMDD`

  Select entries from the log only for the given date.

- `--host=host_name` [350], `-h host_name`

  Connect to the MySQL server on the given host.

- `--password=password` [350], `-p password`

The password to use when connecting to the server.

Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”.

- `--printerror=1` [351], `-e 1`

  Enable error output.

- `--socket=path` [351], `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--user=user_name` [351], `-u user_name`

  The MySQL user name to use when connecting to the server.

4.6.13 mysql_find_rows — Extract SQL Statements from Files

`mysql_find_rows` reads files containing SQL statements and extracts statements that match a given regular expression or that contain `USE db_name` or `SET` statements. The utility was written for use with update log files and as such expects statements to be terminated with semicolon (`;`) characters. It may be useful with other files that contain SQL statements as long as statements are terminated with semicolons.

Invoke `mysql_find_rows` like this:

```
shell> mysql_find_rows [options] [file_name ...]
```

Each *`file_name`* argument should be the name of file containing SQL statements. If no file names are given, `mysql_find_rows` reads the standard input.

Examples:

```
mysql_find_rows --regexp=problem_table --rows=20 < update.log
mysql_find_rows --regexp=problem_table  update-log.1 update-log.2
```

`mysql_find_rows` supports the following options:

- `--help` [351], `--Information` [351]

  Display a help message and exit.

- `--regexp=pattern` [351]

  Display queries that match the pattern.

- `--rows=N` [351]

  Quit after displaying *`N`* queries.

- `--skip-use-db` [351]

  Do not include `USE db_name` statements in the output.

- `--start_row=N` [351]

Start output from this row.

4.6.14 mysql_fix_extensions — Normalize Table File Name Extensions

mysql_fix_extensions converts the extensions for MyISAM (or ISAM) table files to their canonical forms. It looks for files with extensions matching any lettercase variant of .frm, .myd, .myi, .isd, and .ism and renames them to have extensions of .frm, .MYD, .MYI, .ISD, and .ISM, respectively. This can be useful after transferring the files from a system with case-insensitive file names (such as Windows) to a system with case-sensitive file names.

Invoke mysql_fix_extensions like this, where *data_dir* is the path name to the MySQL data directory.

```
shell> mysql_fix_extensions data_dir
```

4.6.15 mysql_setpermission — Interactively Set Permissions in Grant Tables

mysql_setpermission is a Perl script that was originally written and contributed by Luuk de Boer. It interactively sets permissions in the MySQL grant tables. mysql_setpermission is written in Perl and requires that the DBI and DBD::mysql Perl modules be installed (see Section 2.14, “Perl Installation Notes”).

Invoke mysql_setpermission like this:

```
shell> mysql_setpermission [options]
```

options should be either --help [352] to display the help message, or options that indicate how to connect to the MySQL server. The account used when you connect determines which permissions you have when attempting to modify existing permissions in the grant tables.

mysql_setpermissions also reads options from the [client] and [perl] groups in the .my.cnf file in your home directory, if the file exists.

mysql_setpermission supports the following options:

- --help [352]

  Display a help message and exit.

- --host=*host_name* [352]

  Connect to the MySQL server on the given host.

- --password=*password* [352]

  The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- --port=*port_num* [352]

The TCP/IP port number to use for the connection.

- `--socket=path` [353]

  For connections to `localhost`, the Unix socket file to use.

- `--user=user_name` [353]

  The MySQL user name to use when connecting to the server.

4.6.16 mysql_tableinfo — Generate Database Metadata

`mysql_tableinfo` creates tables and populates them with database metadata. It uses `SHOW DATABASES`, `SHOW TABLES`, `SHOW TABLE STATUS`, `SHOW COLUMNS`, and `SHOW INDEX` to obtain the metadata.

Invoke `mysql_tableinfo` like this:

```
shell> mysql_tableinfo [options] db_name [db_like [tbl_like]]
```

The *db_name* argument indicates which database `mysql_tableinfo` should use as the location for the metadata tables. The database will be created if it does not exist. The tables will be named `db`, `tbl` (or `tbl_status`), `col`, and `idx`.

If the *db_like* or *tbl_like* arguments are given, they are used as patterns and metadata is generated only for databases or tables that match the patterns. These arguments default to `%` if not given.

Examples:

```
mysql_tableinfo info
mysql_tableinfo info world
mysql_tableinfo info mydb tmp%
```

Each of the commands stores information into tables in the `info` database. The first stores information for all databases and tables. The second stores information for all tables in the `world` database. The third stores information for tables in the `mydb` database that have names matching the pattern `tmp%`.

Table 4.13 mysql_tableinfo Options

Format	Option File	Description
--clear [354]	clear [354]	Before populating each metadata table, drop it if it exists
--clear-only [354]	clear-only [354]	Similar to --clear, but exits after dropping the metadata tables to be populated.
--col [354]	col [354]	Generate column metadata into the col table
--help [354]		Display help message and exit
--host=host_name [354]	host [354]	Connect to the MySQL server on the given host
--idx [354]	idx [354]	Generate index metadata into the idx table
--password=password [354]	password [354]	The password to use when connecting to the server -- not optional
--port=port_num [354]	port [354]	The TCP/IP port number to use for the connection

Format	Option File	Description
--prefix=prefix_str [354]	prefix [354]	Add prefix_str at the beginning of each metadata table name
--quiet [354]	quiet [354]	Be silent except for errors
--socket=path [354]	socket [354]	Display version information and exit
--tbl-status [355]	tbl-status [355]	Use SHOW TABLE STATUS instead of SHOW TABLES
--user=user_name, [355]	user [355]	The mysql_tableinfo user name to use when connecting to the server

`mysql_tableinfo` supports the following options:

- `--help` [354]

  Display a help message and exit.

- `--clear` [354]

  Before populating each metadata table, drop it if it exists.

- `--clear-only` [354]

  Similar to `--clear` [354], but exits after dropping the metadata tables to be populated.

- `--col` [354]

  Generate column metadata into the `col` table.

- `--host=host_name` [354], `-h host_name`

  Connect to the MySQL server on the given host.

- `--idx` [354]

  Generate index metadata into the `idx` table.

- `--password=password` [354], `-ppassword`

  The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--port=port_num` [354], `-P port_num`

  The TCP/IP port number to use for the connection.

- `--prefix=prefix_str` [354]

  Add *`prefix_str`* at the beginning of each metadata table name.

- `--quiet` [354], `-q`

  Be silent except for errors.

- `--socket=path` [354], `-S path`

The Unix socket file to use for the connection.

- `--tbl-status` [355]

  Use `SHOW TABLE STATUS` instead of `SHOW TABLES`. This provides more complete information, but is slower.

- `--user=user_name` [355], `-u user_name`

  The MySQL user name to use when connecting to the server.

4.6.17 mysql_waitpid — Kill Process and Wait for Its Termination

`mysql_waitpid` signals a process to terminate and waits for the process to exit. It uses the `kill()` system call and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_waitpid` like this:

```
shell> mysql_waitpid [options] pid wait_time
```

`mysql_waitpid` sends signal 0 to the process identified by *`pid`* and waits up to *`wait_time`* seconds for the process to terminate. *`pid`* and *`wait_time`* must be positive integers.

If process termination occurs within the wait time or the process does not exist, `mysql_waitpid` returns 0. Otherwise, it returns 1.

If the `kill()` system call cannot handle signal 0, `mysql_waitpid()` uses signal 1 instead.

`mysql_waitpid` supports the following options:

- `--help` [355], `-?`, `-I`

  Display a help message and exit.

- `--verbose` [355], `-v`

  Verbose mode. Display a warning if signal 0 could not be used and signal 1 is used instead.

- `--version` [355], `-V`

  Display version information and exit.

4.6.18 mysql_zap — Kill Processes That Match a Pattern

`mysql_zap` kills processes that match a pattern. It uses the `ps` command and Unix signals, so it runs on Unix and Unix-like systems.

Invoke `mysql_zap` like this:

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

A process matches if its output line from the `ps` command contains the pattern. By default, `mysql_zap` asks for confirmation for each process. Respond `y` to kill the process, or `q` to exit `mysql_zap`. For any other response, `mysql_zap` does not attempt to kill the process.

If the *-signal* option is given, it specifies the name or number of the signal to send to each process. Otherwise, mysql_zap tries first with TERM (signal 15) and then with KILL (signal 9).

mysql_zap supports the following additional options:

- --help [356], -?, -I

  Display a help message and exit.

- -f

  Force mode. mysql_zap attempts to kill each process without confirmation.

- -t

  Test mode. Display information about each process but do not kill it.

4.7 MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

In shell scripts, you can use the my_print_defaults program to parse option files and see what options would be used by a given program. The following example shows the output that my_print_defaults might produce when asked to show the options found in the [client] and [mysql] groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.7.1 msql2mysql — Convert mSQL Programs for Use with MySQL

Initially, the MySQL C API was developed to be very similar to that for the mSQL database system. Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of the C API functions.

The msql2mysql utility performs the conversion of mSQL C API function calls to their MySQL equivalents. msql2mysql converts the input file in place, so make a copy of the original before converting it. For example, use msql2mysql like this:

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

Then examine client-prog.c and make any post-conversion revisions that may be necessary.

msql2mysql uses the replace utility to make the function name substitutions. See Section 4.8.2, “replace — A String-Replacement Utility”.

4.7.2 mysql_config — Display Options for Compiling Clients

mysql_config provides you with useful information for compiling your MySQL client and connecting it to MySQL. It is a shell script, so it is available only on Unix and Unix-like systems.

mysql_config supports the following options.

- --cflags [357]

  Compiler flags to find include files and critical compiler flags and defines used when compiling the libmysqlclient library. The options returned are tied to the specific compiler that was used when the library was created and might clash with the settings for your own compiler. Use --include [357] for more portable options that contain only include paths.

- --include [357]

  Compiler options to find MySQL include files.

- --libmysqld-libs [357], --embedded [357]

  Libraries and options required to link with the MySQL embedded server.

- --libs [357]

  Libraries and options required to link with the MySQL client library.

- --libs_r [357]

  Libraries and options required to link with the thread-safe MySQL client library.

- --port [357]

  The default TCP/IP port number, defined when configuring MySQL.

- --socket [357]

  The default Unix socket file, defined when configuring MySQL.

- --version [357]

  Version number for the MySQL distribution.

If you invoke mysql_config with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
  --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --include        [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
                    -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lpthread -lz -lcrypt -lnsl -lm -lpthread]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
```

```
  --version        [4.0.16]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz
                    -lcrypt -lnsl -lm -lpthread -lrt]
```

You can use mysql_config within a command line using backticks to include the output that it produces for a particular option. For example, to compile and link a MySQL client program, use mysql_config as follows:

```
shell> gcc -c `mysql_config --cflags` progname.c
shell> gcc -o progname progname.o `mysql_config --libs`
```

4.7.3 my_print_defaults — Display Options from Option Files

my_print_defaults displays the options that are present in option groups of option files. The output indicates what options will be used by programs that read the specified option groups. For example, the mysqlcheck program reads the [mysqlcheck] and [client] option groups. To see what options are present in those groups in the standard option files, invoke my_print_defaults like this:

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=secret
--host=localhost
```

The output consists of options, one per line, in the form that they would be specified on the command line.

my_print_defaults supports the following options.

- --help [358], -?

  Display a help message and exit.

- --config-file=*file_name* [358], --defaults-file=*file_name* [358], -c *file_name*

  Read only the given option file.

- --debug=*debug_options* [358], -# *debug_options*

  Write a debugging log. A typical *debug_options* string is 'd:t:o,*file_name*'. The default is 'd:t:o,/tmp/my_print_defaults.trace'.

- --defaults-extra-file=*file_name* [358], --extra-file=*file_name* [358], -e *file_name*

  Read this option file after the global option file but (on Unix) before the user option file.

- --defaults-group-suffix=*suffix* [358], -g *suffix*

  In addition to the groups named on the command line, read groups that have the given suffix.

- --no-defaults [235], -n

  Return an empty string.

- --verbose [358], -v

  Verbose mode. Print more information about what the program does.

- --version [358], -V

Display version information and exit.

4.7.4 resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols

resolve_stack_dump resolves a numeric stack dump to symbols.

Invoke resolve_stack_dump like this:

```
shell> resolve_stack_dump [options] symbols_file [numeric_dump_file]
```

The symbols file should include the output from the nm --numeric-sort mysqld command. The numeric dump file should contain a numeric stack track from mysqld. If no numeric dump file is named on the command line, the stack trace is read from the standard input.

resolve_stack_dump supports the following options.

- --help [359], -h

  Display a help message and exit.

- --numeric-dump-file=*file_name* [359], -n *file_name*

  Read the stack trace from the given file.

- --symbols-file=*file_name* [359], -s *file_name*

  Use the given symbols file.

- --version [359], -V

  Display version information and exit.

4.8 Miscellaneous Programs

4.8.1 perror — Explain Error Codes

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by examining the documentation for your system or by using the perror utility.

perror prints a description for a system error code or for a storage engine (table handler) error code.

Invoke perror like this:

```
shell> perror [options] errorcode ...
```

Example:

```
shell> perror 13 64
OS error code  13:  Permission denied
OS error code  64:  Machine is not on the network
```

To obtain the error message for a MySQL Cluster error code, invoke perror with the --ndb [360] option:

```
shell> perror --ndb errorcode
```

Note that the meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

perror supports the following options.

- --help [360], --info [360], -I, -?

  Display a help message and exit.

- --ndb [360]

  Print the error message for a MySQL Cluster error code.

- --silent [360], -s

  Silent mode. Print only the error message.

- --verbose [360], -v

  Verbose mode. Print error code and message. This is the default behavior.

- --version [360], -V

  Display version information and exit.

4.8.2 replace — A String-Replacement Utility

The replace utility program changes strings in place in files or on the standard input.

Invoke replace in one of the following ways:

```
shell> replace from to [from to] ... -- file_name [file_name] ...
shell> replace from to [from to] ... < file_name
```

from represents a string to look for and *to* represents its replacement. There can be one or more pairs of strings.

Use the -- option to indicate where the string-replacement list ends and the file names begin. In this case, any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. *replace* prints a message indicating which of the input files it actually modifies.

If the -- option is not given, replace reads the standard input and writes to the standard output.

replace uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps a and b in the given files, file1 and file2:

```
```

```
shell> replace a b b a -- file1 file2 ...
```

The replace program is used by msql2mysql. See Section 4.7.1, “msql2mysql — Convert mSQL Programs for Use with MySQL”.

replace supports the following options.

- -?, -I

  Display a help message and exit.

- -#debug_options

  Enable debugging.

- -s

  Silent mode. Print less information what the program does.

- -v

  Verbose mode. Print more information about what the program does.

- -V

  Display version information and exit.

4.8.3 resolveip — Resolve Host name to IP Address or Vice Versa

The resolveip utility resolves host names to IP addresses and vice versa.

Invoke resolveip like this:

```
shell> resolveip [options] {host_name|ip-addr} ...
```

resolveip supports the following options.

- --help [361], --info [361], -?, -I

  Display a help message and exit.

- --silent [361], -s

  Silent mode. Produce less output.

- --version [361], -V

  Display version information and exit.

Chapter 5 MySQL Server Administration

Table of Contents

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

MySQL Server (`mysqld`) is the main program that does most of the work in a MySQL installation. This section provides an overview of MySQL Server and covers topics that deal with administering a MySQL installation:

- Server configuration
- The server log files
- Security issues and user-account management
- Management of multiple servers on a single machine

5.1 The MySQL Server

`mysqld` is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports
- Server system variables
- Server status variables
- How to set the server SQL mode
- The server shutdown process

Note

Not all storage engines (also known in older versions of MySQL as “table types”) are supported by all MySQL server binaries and configurations. To find out how to determine which storage engines your MySQL server installation supports, see Section 12.4.5.10, “`SHOW ENGINES` Syntax”.

5.1.1 Server Option and Variable Reference

The following table provides a list of all the command line options, server and status variables applicable within `mysqld`.

For a version of this table that is specific to MySQL Cluster, see Section 15.3.4.1, “MySQL Cluster Server Option and Variable Reference”.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with notification of where each option/variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding server system or status variable, the variable name is noted immediately below the corresponding option. For status variables, the scope of the variable is shown (Scope) as either global, session, or both. Please see the corresponding sections for details on setting and using the options and variables. Where appropriate, a direct link to further information on the item as available.

Table 5.1 Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
abort-slave-event-count [1173]	Yes	Yes				
Aborted_clients [449]				Yes	Global	No
Aborted_connects [449]				Yes	Global	No
allow-suspicious-udfs [384]	Yes	Yes				
ansi [384]	Yes	Yes				
autocommit [406]	Yes	Yes	Yes		Session	Yes
back_log [406]			Yes		Global	No
basedir [384]	Yes	Yes	Yes		Global	No
bdb_cache_size [406]			Yes		Global	No
bdb-home [1138]	Yes	Yes			Global	No
- *Variable*: bdb_home [407]			Yes		Global	No
bdb-lock-detect [1138]	Yes	Yes	Yes		Global	No
bdb_log_buffer_size [407]			Yes		Global	No
bdb-logdir [1138]	Yes	Yes			Global	No
- *Variable*: bdb_logdir [407]			Yes		Global	No
bdb_max_lock [407]			Yes		Global	No
bdb-no-recover [1138]	Yes	Yes				
bdb-no-sync [1138]	Yes	Yes				
bdb-shared-data [1138]	Yes	Yes			Global	No
- *Variable*: bdb_shared_data [407]			Yes		Global	No
bdb-tmpdir [1138]	Yes	Yes			Global	No
- *Variable*: bdb_tmpdir [407]			Yes		Global	No
big-tables [384]	Yes	Yes			Session	Yes
- *Variable*: big_tables [407]			Yes		Session	Yes
bind-address [384]	Yes	Yes	Yes		Global	No
Binlog_cache_disk_use [449]				Yes	Global	No
binlog_cache_size [407]	Yes	Yes	Yes		Global	Yes
Binlog_cache_use [449]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
binlog-do-db [1182]	Yes	Yes				
binlog-ignore-db [1182]	Yes	Yes				
bootstrap [384]	Yes	Yes				
bulk_insert_buffer_size [408]	Yes	Yes	Yes		Both	Yes
Bytes_received [449]				Yes	Both	No
Bytes_sent [450]				Yes	Both	No
character_set [408]	Yes		Yes		Global	No
character_set_client [408]			Yes		Both	Yes
character-set-client-handshake [385]	Yes	Yes				
character_set_connection [408]			Yes		Both	Yes
character_set_database [408][a]			Yes		Both	Yes
character_set_results [408]			Yes		Both	Yes
character-set-server [385]	Yes	Yes			Both	Yes
- *Variable*: character_set_server [408]			Yes		Both	Yes
character_set_system [409]			Yes		Global	No
character-sets-dir [385]	Yes	Yes			Global	No
- *Variable*: character_sets_dir [409]			Yes		Global	No
chroot [385]	Yes	Yes				
collation_connection [409]			Yes		Both	Yes
collation_database [409][b]			Yes		Both	Yes
collation-server [385]	Yes	Yes			Both	Yes
- *Variable*: collation_server [409]			Yes		Both	Yes
Com_admin_commands [450]				Yes	Both	No
Com_alter_db [450]				Yes	Both	No
Com_alter_table [450]				Yes	Both	No
Com_analyze [450]				Yes	Both	No
Com_backup_table [450]				Yes	Both	No
Com_begin [450]				Yes	Both	No
Com_change_db [450]				Yes	Both	No
Com_change_master [450]				Yes	Both	No
Com_check [450]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_checksum [450]				Yes	Both	No
Com_commit [450]				Yes	Both	No
Com_create_db [450]				Yes	Both	No
Com_create_index [450]				Yes	Both	No
Com_create_table [450]				Yes	Both	No
Com_dealloc_sql [450]				Yes	Both	No
Com_delete [450]				Yes	Both	No
Com_delete_multi [450]				Yes	Both	No
Com_do [450]				Yes	Both	No
Com_drop_db [450]				Yes	Both	No
Com_drop_index [450]				Yes	Both	No
Com_drop_table [450]				Yes	Both	No
Com_drop_user [450]				Yes	Both	No
Com_execute_sql [450]				Yes	Both	No
Com_flush [450]				Yes	Both	No
Com_grant [450]				Yes	Both	No
Com_ha_close [450]				Yes	Both	No
Com_ha_open [450]				Yes	Both	No
Com_ha_read [450]				Yes	Both	No
Com_help [450]				Yes	Both	No
Com_insert [450]				Yes	Both	No
Com_insert_select [450]				Yes	Both	No
Com_kill [450]				Yes	Both	No
Com_load [450]				Yes	Both	No
Com_load_master_data [450]				Yes	Both	No
Com_load_master_table [450]				Yes	Both	No
Com_lock_tables [450]				Yes	Both	No
Com_optimize [450]				Yes	Both	No
Com_preload_keys [450]				Yes	Both	No
Com_prepare_sql [450]				Yes	Both	No
Com_rename_table [450]				Yes	Both	No
Com_repair [450]				Yes	Both	No
Com_replace [450]				Yes	Both	No
Com_replace_select [450]				Yes	Both	No
Com_reset [450]				Yes	Both	No
Com_restore_table [450]				Yes	Both	No
Com_revoke [450]				Yes	Both	No
Com_revoke_all [450]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_rollback [450]				Yes	Both	No
Com_savepoint [450]				Yes	Both	No
Com_select [450]				Yes	Both	No
Com_set_option [450]				Yes	Both	No
Com_show_binlog_events [450]				Yes	Both	No
Com_show_binlogs [450]				Yes	Both	No
Com_show_charsets [450]				Yes	Both	No
Com_show_collations [450]				Yes	Both	No
Com_show_column_types [450]				Yes	Both	No
Com_show_create_db [450]				Yes	Both	No
Com_show_create_event [450]				Yes	Both	No
Com_show_create_table [450]				Yes	Both	No
Com_show_databases [450]				Yes	Both	No
Com_show_engine_logs [450]				Yes	Both	No
Com_show_engine_mutex [450]				Yes	Both	No
Com_show_engine_status [450]				Yes	Both	No
Com_show_errors [450]				Yes	Both	No
Com_show_fields [450]				Yes	Both	No
Com_show_grants [450]				Yes	Both	No
Com_show_innodb_status [450]				Yes	Both	No
Com_show_keys [450]				Yes	Both	No
Com_show_logs [450]				Yes	Both	No
Com_show_master_status [450]				Yes	Both	No
Com_show_ndb_status [450]				Yes	Both	No
Com_show_new_master [450]				Yes	Both	No
Com_show_open_tables [450]				Yes	Both	No
Com_show_privileges [450]				Yes	Both	No
Com_show_processlist [450]				Yes	Both	No
Com_show_slave_hosts [450]				Yes	Both	No
Com_show_slave_status [450]				Yes	Both	No
Com_show_status [450]				Yes	Both	No
Com_show_storage_engines [450]				Yes	Both	No
Com_show_tables [450]				Yes	Both	No
Com_show_variables [450]				Yes	Both	No
Com_show_warnings [450]				Yes	Both	No
Com_slave_start [450]				Yes	Both	No
Com_slave_stop [450]				Yes	Both	No
Com_stmt_close [450]				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_stmt_execute [450]				Yes	Both	No
Com_stmt_fetch [450]				Yes	Both	No
Com_stmt_prepare [450]				Yes	Both	No
Com_stmt_reset [450]				Yes	Both	No
Com_stmt_send_long_data [450]				Yes	Both	No
Com_truncate [450]				Yes	Both	No
Com_unlock_tables [450]				Yes	Both	No
Com_update [450]				Yes	Both	No
Com_update_multi [450]				Yes	Both	No
concurrent_insert [406]	Yes	Yes	Yes		Global	Yes
connect_timeout [409]	Yes	Yes	Yes		Global	Yes
console [385]	Yes	Yes				
core-file [385]	Yes	Yes				
Created_tmp_disk_tables [450]				Yes	Both	No
Created_tmp_files [451]				Yes	Global	No
Created_tmp_tables [451]				Yes	Both	No
datadir [385]	Yes	Yes	Yes		Global	No
date_format [409]			Yes		Both	No
datetime_format [410]			Yes		Both	No
debug [385]	Yes	Yes	Yes		Both	Yes
default-character-set [385]	Yes	Yes				
default-collation [386]	Yes	Yes				
default-storage-engine [386]	Yes	Yes	Yes		Both	Yes
default-table-type [386]	Yes	Yes				
default-time-zone [386]	Yes	Yes				
default_week_format [410]	Yes	Yes	Yes		Both	Yes
defaults-extra-file [235]	Yes					
defaults-file [235]	Yes					
delay-key-write [386]	Yes	Yes			Global	Yes
- *Variable*: delay_key_write [410]			Yes		Global	Yes
delayed_insert_limit [410]	Yes	Yes	Yes		Global	Yes
delayed_insert_timeout [410]	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
delayed_queue_size [410]	Yes	Yes	Yes		Global	Yes
des-key-file [386]	Yes	Yes				
disconnect-slave-event-count [1173]	Yes	Yes				
enable-locking [386]	Yes	Yes				
enable-pstack [387]	Yes	Yes				
error_count [410]			Yes		Session	No
exit-info [387]	Yes	Yes				
expire_logs_days [411]	Yes	Yes	Yes		Global	Yes
external-locking [387]	Yes	Yes				
- *Variable*: skip_external_locking [427]						
flush [411]	Yes	Yes	Yes		Global	Yes
flush_time [411]	Yes	Yes	Yes		Global	Yes
foreign_key_checks [411]			Yes		Session	Yes
ft_boolean_syntax [411]	Yes	Yes	Yes		Global	Yes
ft_max_word_len [412]	Yes	Yes	Yes		Global	No
ft_min_word_len [412]	Yes	Yes	Yes		Global	No
ft_query_expansion_limit [412]	Yes	Yes	Yes		Global	No
ft_stopword_file [412]	Yes	Yes	Yes		Global	No
gdb [387]	Yes	Yes				
group_concat_max_len [412]	Yes	Yes	Yes		Both	Yes
Handler_commit [451]				Yes	Both	No
Handler_delete [451]				Yes	Both	No
Handler_discover [1306]				Yes	Both	No
Handler_read_first [451]				Yes	Both	No
Handler_read_key [451]				Yes	Both	No
Handler_read_next [451]				Yes	Both	No
Handler_read_prev [451]				Yes	Both	No
Handler_read_rnd [452]				Yes	Both	No
Handler_read_rnd_next [452]				Yes	Both	No
Handler_rollback [452]				Yes	Both	No
Handler_update [452]				Yes	Both	No
Handler_write [452]				Yes	Both	No
have_archive [412]			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
have_bdb [412]			Yes		Global	No
have_blackhole_engine [413]			Yes		Global	No
have_compress [413]			Yes		Global	No
have_crypt [413]			Yes		Global	No
have_csv [413]			Yes		Global	No
have_example_engine [413]			Yes		Global	No
have_geometry [413]			Yes		Global	No
have_innodb [413]			Yes		Global	No
have_isam [413]			Yes		Global	No
have_merge_engine [413]			Yes		Global	No
have_ndbcluster [1302]			Yes		Global	No
have_openssl [413]			Yes		Global	No
have_query_cache [413]			Yes		Global	No
have_raid [413]			Yes		Global	No
have_rtree_keys [413]			Yes		Global	No
have_symlink [413]			Yes		Global	No
help [384]	Yes	Yes				
identity [414]			Yes		Session	Yes
init_connect [414]	Yes	Yes	Yes		Global	Yes
init-file [387]	Yes	Yes			Global	No
- *Variable*: init_file [414]			Yes		Global	No
init-rpl-role	Yes	Yes				
init_slave [1180]	Yes	Yes	Yes		Global	Yes
innodb [1066]	Yes	Yes				
innodb_additional_mem_pool_size [1066]	Yes	Yes	Yes		Global	No
innodb_autoextend_increment [1067]	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_awe_mem_mb [1067]	Yes	Yes	Yes		Global	No
innodb_buffer_pool_size [1067]	Yes	Yes	Yes		Global	No
innodb_data_file_path [1067]	Yes	Yes	Yes		Global	No
innodb_data_home_dir [1068]	Yes	Yes	Yes		Global	No
innodb_fast_shutdown [1068]	Yes	Yes	Yes		Global	Yes
innodb_file_io_threads [1068]	Yes	Yes	Yes		Global	No
innodb_file_per_table [1068]	Yes	Yes	Yes		Global	No
innodb_flush_log_at_trx_commit [1068]	Yes	Yes	Yes		Global	Yes
innodb_flush_method [1069]	Yes	Yes	Yes		Global	No
innodb_force_recovery [1069]	Yes	Yes	Yes		Global	No
innodb_lock_wait_timeout [1069]	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_locks_unsafe_for_binlog [1070]	Yes	Yes	Yes		Global	No
innodb_log_arch_dir [1071]	Yes	Yes	Yes		Global	No
innodb_log_archive [1071]	Yes	Yes	Yes		Global	No
innodb_log_buffer_size [1071]	Yes	Yes	Yes		Global	No
innodb_log_file_size [1071]	Yes	Yes	Yes		Global	No
innodb_log_files_in_group [1071]	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir [1071]	Yes	Yes	Yes		Global	No
innodb_max_dirty_pages_pct [1071]	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag [1071]	Yes	Yes	Yes		Global	Yes
innodb_mirrored_log_groups [1072]	Yes	Yes	Yes		Global	No
innodb_open_files [1072]	Yes	Yes	Yes		Global	No
innodb-safe-binlog [387]	Yes	Yes				
innodb-status-file [1066]	Yes	Yes				
innodb_table_locks [1072]	Yes	Yes	Yes		Both	Yes
innodb_thread_concurrency [1072]	Yes	Yes	Yes		Global	Yes
insert_id [414]			Yes		Session	Yes
install [387]	Yes					
install-manual [387]	Yes					
interactive_timeout [414]	Yes	Yes	Yes		Both	Yes
isam	Yes	Yes				
join_buffer_size [415]	Yes	Yes	Yes		Both	Yes
Key_blocks_not_flushed [452]				Yes	Global	No
Key_blocks_unused [452]				Yes	Global	No
Key_blocks_used [452]				Yes	Global	No
key_buffer_size [415]	Yes	Yes	Yes		Global	Yes
key_cache_age_threshold [416]	Yes	Yes	Yes		Global	Yes
key_cache_block_size [416]	Yes	Yes	Yes		Global	Yes
key_cache_division_limit [416]	Yes	Yes	Yes		Global	Yes
Key_read_requests [452]				Yes	Global	No
Key_reads [452]				Yes	Global	No
Key_write_requests [452]				Yes	Global	No
Key_writes [452]				Yes	Global	No
language [388]	Yes	Yes	Yes		Global	No
last_insert_id [416]			Yes		Session	Yes
lc_time_names [417]			Yes		Both	Yes
license [417]			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
local_infile [417]			Yes		Global	Yes
local-service	Yes					
locked_in_memory [417]			Yes		Global	No
log [388]	Yes	Yes	Yes		Global	No
log_bin [1183]			Yes		Global	No
log-bin [1181]	Yes	Yes	Yes		Global	No
log-bin-index [1182]	Yes	Yes				
log-error [388]	Yes	Yes			Global	No
- *Variable*: log_error [417]			Yes		Global	No
log-isam [388]	Yes	Yes				
log-long-format [388]	Yes	Yes				
log-queries-not-using-indexes [388]	Yes	Yes			Global	Yes
- *Variable*: log_queries_not_using_indexes			Yes		Global	Yes
log-short-format [388]	Yes	Yes				
log-slave-updates [1173]	Yes	Yes			Global	No
- *Variable*: log_slave_updates [1183]			Yes		Global	No
log_slave_updates [1183]	Yes	Yes	Yes		Global	No
log-slow-admin-statements [388]	Yes	Yes				
log-slow-queries [388]	Yes	Yes			Global	No
- *Variable*: log_slow_queries [417]			Yes		Global	No
log-update [389]	Yes					
- *Variable*: log_update [417]						
log-warnings [389]	Yes	Yes			Both	Yes
- *Variable*: log_warnings [417]			Yes		Both	Yes
long_query_time [417]	Yes	Yes	Yes		Both	Yes
low-priority-updates [389]	Yes	Yes			Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
- *Variable*: low_priority_updates [418]			Yes		Both	Yes
lower_case_file_system [418]			Yes		Global	No
lower_case_table_names [418]	Yes	Yes	Yes		Global	No
master-connect-retry [1173]	Yes	Yes				
master-host [1174]	Yes	Yes				
master-info-file [1174]	Yes	Yes				
master-password [1174]	Yes	Yes				
master-port [1174]	Yes	Yes				
master-retry-count [1174]	Yes	Yes				
master-ssl [1174]	Yes	Yes				
master-ssl-ca [1174]	Yes	Yes				
master-ssl-capath [1174]	Yes	Yes				
master-ssl-cert [1174]	Yes	Yes				
master-ssl-cipher [1174]	Yes	Yes				
master-ssl-key [1174]	Yes	Yes				
master-user [1174]	Yes	Yes				
max_allowed_packet [418]	Yes	Yes	Yes		Global	Yes
max_binlog_cache_size [1183]	Yes	Yes	Yes		Global	Yes
max-binlog-dump-events [1183]	Yes	Yes				
max_binlog_size [1184]	Yes	Yes	Yes		Global	Yes
max_connect_errors [419]	Yes	Yes	Yes		Global	Yes
max_connections [416]	Yes	Yes	Yes		Global	Yes
max_delayed_threads [419]	Yes	Yes	Yes		Both	Yes
max_error_count [420]	Yes	Yes	Yes		Both	Yes
max_heap_table_size [419]	Yes	Yes	Yes		Both	Yes
max_insert_delayed_threads [419]			Yes		Both	Yes
max_join_size [419]	Yes	Yes	Yes		Both	Yes
max_length_for_sort_data [420]	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
max_prepared_stmt_count [420]	Yes	Yes	Yes		Global	Yes
max_relay_log_size [420]	Yes	Yes	Yes		Global	Yes
max_seeks_for_key [420]	Yes	Yes	Yes		Both	Yes
max_sort_length [420]	Yes	Yes	Yes		Both	Yes
Max_used_connections [453]				Yes	Global	No
max_user_connections [420]	Yes	Yes	Yes		Global	Yes
max_write_lock_count [421]	Yes	Yes	Yes		Global	Yes
memlock [389]	Yes	Yes	Yes		Global	No
merge	Yes	Yes				
myisam-block-size [390]	Yes	Yes				
myisam_data_pointer_size [421]	Yes	Yes	Yes		Global	Yes
myisam_max_extra_sort_file_size [421]	Yes	Yes	Yes		Global	No
myisam_max_sort_file_size [421]	Yes	Yes	Yes		Global	Yes
myisam-recover [390]	Yes	Yes				
- *Variable*: myisam_recover_options [421]						
myisam_recover_options [421]			Yes		Global	No
myisam_repair_threads [421]	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size [421]	Yes	Yes	Yes		Both	Yes
myisam_stats_method [422]	Yes	Yes	Yes		Both	Yes
named_pipe [422]			Yes		Global	No
ndb_autoincrement_prefetch_sz [1302]	Yes	Yes	Yes		Both	Yes
ndb_cache_check_time [1302]	Yes	Yes	Yes		Global	Yes
ndb_force_send [1302]	Yes	Yes	Yes		Both	Yes
ndb_index_stat_cache_entries [1302]	Yes	Yes				
ndb_index_stat_enable [1303]	Yes	Yes				
ndb_index_stat_update_freq [1304]	Yes	Yes				
ndb_optimized_node_selection [1304]	Yes	Yes				
ndb_report_thresh_binlog_epoch_slip [1304]	Yes	Yes				
ndb_report_thresh_binlog_mem_usage [1305]	Yes	Yes				
ndb_use_exact_count [1305]			Yes		Both	Yes
ndb_use_transactions [1305]	Yes	Yes	Yes		Both	Yes
ndbcluster [1301]	Yes	Yes				
- *Variable*: have_ndbcluster [1302]						
net_buffer_length [422]	Yes	Yes	Yes		Both	Yes
net_read_timeout [422]	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
net_retry_count [422]	Yes	Yes	Yes		Both	Yes
net_write_timeout [422]	Yes	Yes	Yes		Both	Yes
new [390]	Yes	Yes	Yes		Both	Yes
no-defaults [235]	Yes					
Not_flushed_delayed_rows				Yes	Global	No
old_passwords [423]			Yes		Both	Yes
old-protocol [391]	Yes	Yes				
Open_files				Yes	Global	No
open-files-limit [391]	Yes	Yes			Global	No
- *Variable*: open_files_limit [423]			Yes		Global	No
Open_streams [453]				Yes	Global	No
Open_tables [453]				Yes	Both	No
Opened_tables [453]				Yes	Both	No
pid-file [391]	Yes	Yes			Global	No
- *Variable*: pid_file [423]			Yes		Global	No
plugin_dir [423]	Yes	Yes	Yes		Global	No
port [391]	Yes	Yes	Yes		Global	No
preload_buffer_size [423]	Yes	Yes	Yes		Both	Yes
prepared_stmt_count [423]			Yes		Both	No
Prepared_stmt_count [453]				Yes	Global	No
print-defaults [235]	Yes					
protocol_version [423]			Yes		Global	No
pseudo_thread_id [424]			Yes		Session	Yes
Qcache_free_blocks [453]				Yes	Global	No
Qcache_free_memory [453]				Yes	Global	No
Qcache_hits [453]				Yes	Global	No
Qcache_inserts [453]				Yes	Global	No
Qcache_lowmem_prunes [453]				Yes	Global	No
Qcache_not_cached [453]				Yes	Global	No
Qcache_queries_in_cache [454]				Yes	Global	No
Qcache_total_blocks [454]				Yes	Global	No
query_alloc_block_size [424]	Yes	Yes	Yes		Both	Yes
query_cache_limit [424]	Yes	Yes	Yes		Global	Yes
query_cache_min_res_unit [424]	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
query_cache_size [424]	Yes	Yes	Yes		Global	Yes
query_cache_type [424]	Yes	Yes	Yes		Both	Yes
query_cache_wlock_invalidate [425]	Yes	Yes	Yes		Both	Yes
query_prealloc_size [425]	Yes	Yes	Yes		Both	Yes
Questions [454]				Yes	Both	No
rand_seed1 [425]			Yes		Session	Yes
rand_seed2 [425]			Yes		Session	Yes
range_alloc_block_size [425]	Yes	Yes	Yes		Both	Yes
read_buffer_size [425]	Yes	Yes	Yes		Both	Yes
read_only [425]	Yes	Yes	Yes		Global	Yes
read_rnd_buffer_size [426]	Yes	Yes	Yes		Both	Yes
relay-log [1175]	Yes	Yes			Global	No
- *Variable*: relay_log			Yes		Global	No
relay-log-index [1175]	Yes	Yes			Global	No
- *Variable*: relay_log_index			Yes		Global	No
relay_log_index	Yes	Yes	Yes		Global	No
relay-log-info-file [1175]	Yes	Yes				
- *Variable*: relay_log_info_file						
relay_log_info_file	Yes	Yes	Yes		Global	No
relay_log_purge [426]	Yes	Yes	Yes		Global	Yes
relay_log_space_limit [426]	Yes	Yes	Yes		Global	No
remove [391]	Yes					
replicate-do-db [1176]	Yes	Yes				
replicate-do-table [1177]	Yes	Yes				
replicate-ignore-db [1176]	Yes	Yes				
replicate-ignore-table [1177]	Yes	Yes				
replicate-rewrite-db [1177]	Yes	Yes				
replicate-same-server-id [1177]	Yes	Yes				
replicate-wild-do-table [1178]	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
replicate-wild-ignore-table [1178]	Yes	Yes				
report-host [1178]	Yes	Yes			Global	No
- *Variable*: report_host			Yes		Global	No
report-password [1178]	Yes	Yes			Global	No
- *Variable*: report_password			Yes		Global	No
report-port [1179]	Yes	Yes			Global	No
- *Variable*: report_port			Yes		Global	No
report-user [1179]	Yes	Yes			Global	No
- *Variable*: report_user			Yes		Global	No
rpl_recovery_rank [1180]			Yes		Global	Yes
safe-mode [391]	Yes	Yes				
safe-show-database [391]	Yes	Yes	Yes		Global	Yes
safe-user-create [392]	Yes	Yes				
safemalloc-mem-limit	Yes	Yes				
secure-auth [392]	Yes	Yes			Global	Yes
- *Variable*: secure_auth [426]			Yes		Global	Yes
Select_full_join [454]				Yes	Both	No
Select_full_range_join [454]				Yes	Both	No
Select_range [454]				Yes	Both	No
Select_range_check [454]				Yes	Both	No
Select_scan [454]				Yes	Both	No
server-id [1167]	Yes	Yes			Global	Yes
- *Variable*: server_id [426]			Yes		Global	Yes
set-variable	Yes	Yes				
shared_memory [426]			Yes		Global	No
shared_memory_base_name [427]			Yes		Global	No
show-slave-auth-info [1179]	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
skip-bdb [392]	Yes	Yes				
skip-character-set-client-handshake [385]	Yes	Yes				
skip-concurrent-insert [392]	Yes	Yes				
- *Variable*: concurrent_insert [409]						
skip_external_locking [427]	Yes	Yes	Yes		Global	No
skip-grant-tables [392]	Yes	Yes				
skip-host-cache [392]	Yes	Yes				
skip-isam	Yes	Yes				
skip-locking [387]	Yes	Yes				
skip-name-resolve [393]	Yes	Yes			Global	No
- *Variable*: skip_name_resolve			Yes		Global	No
skip-networking [393]	Yes	Yes			Global	No
- *Variable*: skip_networking [427]			Yes		Global	No
skip-new	Yes	Yes				
skip-safemalloc [394]	Yes	Yes				
skip-show-database [394]	Yes	Yes			Global	No
- *Variable*: skip_show_database [427]			Yes		Global	No
skip-slave-start [1179]	Yes	Yes				
skip-ssl [521]	Yes	Yes				
skip-stack-trace [394]	Yes	Yes				
skip-symbolic-links [393]	Yes					
skip-symlink [393]	Yes	Yes				
skip-sync-bdb-logs [1138]	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
skip-thread-priority [394]	Yes	Yes				
slave_compressed_protocol [1180]	Yes	Yes	Yes		Global	Yes
slave-load-tmpdir [1179]	Yes	Yes			Global	No
- *Variable*: slave_load_tmpdir [1180]			Yes		Global	No
slave-net-timeout [1179]	Yes	Yes			Global	Yes
- *Variable*: slave_net_timeout [1181]			Yes		Global	Yes
Slave_open_temp_tables [454]				Yes	Global	No
slave-skip-errors [1179]	Yes	Yes			Global	No
- *Variable*: slave_skip_errors [1181]			Yes		Global	No
slave_transaction_retries [1181]	Yes	Yes	Yes		Global	Yes
Slow_launch_threads [455]				Yes	Both	No
slow_launch_time [427]	Yes	Yes	Yes		Global	Yes
Slow_queries [455]				Yes	Both	No
socket [394]	Yes	Yes	Yes		Global	No
sort_buffer_size [427]	Yes	Yes	Yes		Both	Yes
Sort_merge_passes [455]				Yes	Both	No
Sort_range [455]				Yes	Both	No
Sort_rows [455]				Yes	Both	No
Sort_scan [455]				Yes	Both	No
sporadic-binlog-dump-fail [1183]	Yes	Yes				
sql_auto_is_null [428]			Yes		Session	Yes
sql_big_selects [428]			Yes		Session	Yes
sql_big_tables [407]			Yes		Session	Yes
sql-bin-update-same	Yes	Yes				
sql_buffer_result [428]			Yes		Session	Yes
sql_log_bin [429]			Yes		Session	Yes
sql_log_off [429]			Yes		Session	Yes
sql_log_update [429]			Yes		Session	Yes
sql_low_priority_updates [418]			Yes		Both	Yes
sql_max_join_size [419]			Yes		Both	Yes
sql-mode [394]	Yes	Yes			Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
- *Variable*: sql_mode [429]			Yes		Both	Yes
sql_notes [429]			Yes		Session	Yes
sql_quote_show_create [430]			Yes		Session	Yes
sql_safe_updates [430]			Yes		Session	Yes
sql_select_limit [430]			Yes		Both	Yes
sql_slave_skip_counter [1181]			Yes		Global	Yes
sql_warnings [430]			Yes		Session	Yes
ssl [521]	Yes	Yes				
Ssl_accept_renegotiates [455]				Yes	Global	No
Ssl_accepts [455]				Yes	Global	No
ssl-ca [522]	Yes	Yes			Global	No
- *Variable*: ssl_ca			Yes		Global	No
Ssl_callback_cache_hits [455]				Yes	Global	No
ssl-capath [522]	Yes	Yes			Global	No
- *Variable*: ssl_capath			Yes		Global	No
ssl-cert [522]	Yes	Yes			Global	No
- *Variable*: ssl_cert			Yes		Global	No
ssl-cipher [522]	Yes	Yes			Global	No
- *Variable*: ssl_cipher			Yes		Global	No
Ssl_cipher [455]				Yes	Both	No
Ssl_cipher_list [455]				Yes	Both	No
Ssl_client_connects [455]				Yes	Global	No
Ssl_connect_renegotiates [455]				Yes	Global	No
Ssl_ctx_verify_depth [456]				Yes	Global	No
Ssl_ctx_verify_mode [456]				Yes	Global	No
Ssl_default_timeout [456]				Yes	Both	No
Ssl_finished_accepts [456]				Yes	Global	No
Ssl_finished_connects [456]				Yes	Global	No
ssl-key [522]	Yes	Yes			Global	No
- *Variable*: ssl_key			Yes		Global	No
Ssl_session_cache_hits [456]				Yes	Global	No
Ssl_session_cache_misses [456]				Yes	Global	No
Ssl_session_cache_mode [456]				Yes	Global	No
Ssl_session_cache_overflows [456]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Ssl_session_cache_size [456]				Yes	Global	No
Ssl_session_cache_timeouts [456]				Yes	Global	No
Ssl_sessions_reused [456]				Yes	Both	No
Ssl_used_session_cache_entries [456]				Yes	Global	No
Ssl_verify_depth [456]				Yes	Both	No
Ssl_verify_mode [456]				Yes	Both	No
Ssl_version [457]				Yes	Both	No
standalone [393]	Yes	Yes				
storage_engine [430]			Yes		Both	Yes
symbolic-links [393]	Yes	Yes				
sync-bdb-logs [1138]	Yes	Yes	Yes		Global	No
sync_binlog [1184]	Yes	Yes	Yes		Global	Yes
sync_frm [430]	Yes	Yes	Yes		Global	Yes
system_time_zone [430]			Yes		Global	No
table_cache [431]	Yes	Yes	Yes		Global	Yes
Table_locks_immediate [457]				Yes	Global	No
Table_locks_waited [457]				Yes	Global	No
table_type [430]			Yes		Both	Yes
temp-pool [394]	Yes	Yes				
thread_cache_size [431]	Yes	Yes	Yes		Global	Yes
thread_concurrency [431]	Yes	Yes	Yes		Global	No
thread_stack [431]	Yes	Yes	Yes		Global	No
Threads_cached [457]				Yes	Global	No
Threads_connected [457]				Yes	Global	No
Threads_created [457]				Yes	Global	No
Threads_running [457]				Yes	Global	No
time_format [431]			Yes		Both	No
time_zone [431]			Yes		Both	Yes
timestamp [432]			Yes		Session	Yes
tmp_table_size [432]	Yes	Yes	Yes		Both	Yes
tmpdir [394]	Yes	Yes	Yes		Global	No
transaction_alloc_block_size [432]	Yes	Yes	Yes		Both	Yes
transaction-isolation [394]	Yes	Yes				
- *Variable*: tx_isolation [433]						
transaction_prealloc_size [432]	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
tx_isolation [433]			Yes		Both	Yes
unique_checks [433]			Yes		Session	Yes
Uptime [457]				Yes	Global	No
user [395]	Yes	Yes				
verbose [395]	Yes	Yes				
version [433]			Yes		Global	No
version_comment [433]			Yes		Global	No
version_compile_machine [434]			Yes		Global	No
version_compile_os [434]			Yes		Global	No
wait_timeout [434]	Yes	Yes	Yes		Both	Yes
warning_count [434]			Yes		Session	No
warnings	Yes	Yes				

[a] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

[b] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

5.1.2 Server Command Options

When you start the `mysqld` server, you can specify program options using any of the methods described in Section 4.2.3, “Specifying Program Options”. The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See Section 4.2.3.3, “Using Option Files”.

`mysqld` reads options from the `[mysqld]` and `[server]` groups. `mysqld_safe` reads options from the `[mysqld]`, `[server]`, `[mysqld_safe]`, and `[safe_mysqld]` groups. `mysql.server` reads options from the `[mysqld]` and `[mysql.server]` groups.

An embedded MySQL server usually reads options from the `[server]`, `[embedded]`, and `[xxxxx_SERVER]` groups, where *xxxxx* is the name of the application into which the server is embedded.

`mysqld` accepts many command options. For a list, execute `mysqld --help`. Before MySQL 4.1.1, `--help` [384] prints the full help message. As of 4.1.1, it prints a brief message; to see the full list, use `mysqld --verbose --help`.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See Section 5.4.4, “Security-Related `mysqld` Options”.
- SSL-related options: See Section 5.6.6.3, “SSL Command Options”.
- Binary log control options: See Section 14.8.4, “Binary Log Options and Variables”.
- Replication-related options: See Section 14.8, “Replication and Binary Logging Options and Variables”.
- Options specific to particular storage engines: See Section 13.1.1, “`MyISAM` Startup Options”, Section 13.5.3, “`BDB` Startup Options”, Section 13.2.4, “`InnoDB` Startup Options and System Variables”, and Section 15.3.4.2, “`mysqld` Command Options for MySQL Cluster”.

You can also set the values of server system variables by using variable names as options, as described at the end of this section.

Some options control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to an option that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to an option for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some options take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued option is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- `--help` [384], `-?`

  Display a short help message and exit. Before MySQL 4.1.1, `--help` [384] displays the full help message. As of 4.1.1, it displays an abbreviated message only. Use both the `--verbose` [395] and `--help` [384] options to see the full message.

- `--allow-suspicious-udfs` [384]

  This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. This option was added in MySQL 4.0.24, and 4.1.10a. See Section 18.2.2.6, “User-Defined Function Security Precautions”.

- `--ansi` [384]

  Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the `--sql-mode` [394] option instead. See Section 1.9.3, “Running MySQL in ANSI Mode”, and Section 5.1.6, “Server SQL Modes”.

- `--basedir=`*`path`* [384], `-b` *`path`* [384]

  The path to the MySQL installation directory. All paths are usually resolved relative to this directory.

- `--big-tables` [384]

  Enable large result sets by saving all temporary sets in files. This option prevents most “table full” errors, but also slows down queries for which in-memory tables would suffice. Since MySQL 3.23.2, the server is able to handle large result sets automatically by using memory for small temporary tables and switching to disk tables where necessary.

- `--bind-address=`*`IP`* [384]

  The IP address to bind to. Only one address can be selected. If this option is specified multiple times, the last address given is used.

  If no address or `0.0.0.0` is specified, the server listens on all interfaces.

- `--bootstrap` [384]

  This option is used by the `mysql_install_db` script to create the MySQL privilege tables without having to start a full MySQL server.

- `--character-sets-dir=path` [385]

  The directory where character sets are installed. See Section 9.6, “Character Set Configuration”.

- `--character-set-client-handshake` [385]

  Do not ignore character set information sent by the client. To ignore client information and use the default server character set, use `--skip-character-set-client-handshake` [385]; this makes MySQL 4.1 and higher behave like MySQL 4.0. This option was added in MySQL 4.1.15.

- `--character-set-server=charset_name` [385], `-C charset_name`

  Use `charset_name` as the default server character set. See Section 9.6, “Character Set Configuration”. If you use this option to specify a nondefault character set, you should also use `--collation-server` [385] to specify the collation. This option is available as of MySQL 4.1.3.

- `--chroot=path` [385], `-r path`

  Put the `mysqld` server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure as of MySQL 4.0. (MySQL 3.23 is not able to provide a `chroot()` jail that is 100% closed.) Note that use of this option somewhat limits `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

- `--collation-server=collation_name` [385]

  Use `collation_name` as the default server collation. This option is available as of MySQL 4.1.3. See Section 9.6, “Character Set Configuration”.

- `--console` [385]

  (Windows only.) Write error log messages to `stderr` and `stdout` even if `--log-error` [388] is specified. `mysqld` does not close the console window if this option is used.

- `--core-file` [385]

  Write a core file if `mysqld` dies. The name and location of the core file is system dependent. On Linux, a core file named `core.pid` is written to the current working directory of the process, which for `mysqld` is the data directory. `pid` represents the process ID of the server process. On Mac OS X, a core file named `core.pid` is written to the `/cores` directory. On Solaris, use the `coreadm` command to specify where to write the core file and how to name it.

  For some systems, to get a core file you must also specify the `--core-file-size` [244] option to `mysqld_safe`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”. On some systems, such as Solaris, you do not get a core file if you are also using the `--user` [395] option. There might be additional restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation.

- `--datadir=path` [385], `-h path`

  The path to the data directory.

- `--debug[=debug_options]` [385], `-# [debug_options]`

  If MySQL is configured with `--with-debug` [98], you can use this option to get a trace file of what `mysqld` is doing. A typical `debug_options` string is `'d:t:o,file_name'`. The default is `'d:t:i:o,mysqld.trace'`.

- `--default-character-set=charset_name` [385], `-C charset_name`

Use *`charset_name`* as the default character set. This option is deprecated in favor of `--character-set-server` [385] as of MySQL 4.1.3. See Section 9.6, “Character Set Configuration”.

For more information, see Section 18.4.3, “The DBUG Package”.

- `--default-collation=`*`collation_name`* [386]

  Use *`collation_name`* as the default collation. This option is deprecated in favor of `--collation-server` [385] as of MySQL 4.1.3. See Section 9.6, “Character Set Configuration”.

- `--default-storage-engine=`*`type`* [386]

  This option is a synonym for `--default-table-type` [386]. It is available as of MySQL 4.1.2.

- `--default-table-type=`*`type`* [386]

  Set the default table type (storage engine) for tables. See Chapter 13, *Storage Engines*.

- `--default-time-zone=`*`timezone`* [386]

  Set the default server time zone. This option sets the global `time_zone` [431] system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` [430] system variable. This option is available as of MySQL 4.1.3.

- `--delay-key-write[={OFF|ON|ALL}]` [386]

  Specify how to use delayed key writes. Delayed key writing causes key buffers not to be flushed between writes for `MyISAM` tables. `OFF` disables delayed key writes. `ON` enables delayed key writes for those tables that were created with the `DELAY_KEY_WRITE` option. `ALL` delays key writes for all `MyISAM` tables. Available as of MySQL 4.0.3. See Section 7.8.2, “Tuning Server Parameters”, and Section 13.1.1, “`MyISAM` Startup Options”.

  Note

  If you set this variable to `ALL`, you should not use `MyISAM` tables from within another program (such as another MySQL server or `myisamchk`) when the tables are in use. Doing so leads to index corruption.

- `--delay-key-write-for-all-tables` [386]

  Old form of `--delay-key-write=ALL` [386] for use prior to MySQL 4.0.3. As of 4.0.3, use `--delay-key-write=ALL` [386] instead. `--delay-key-write-for-all-tables` [386] is removed in MySQL 5.5.

- `--des-key-file=`*`file_name`* [386]

  Read the default DES keys from this file. These keys are used by the `DES_ENCRYPT()` [866] and `DES_DECRYPT()` [866] functions.

- `--enable-locking` [386]

  This option is deprecated. Use `--external-locking` [387] instead.

- `--enable-named-pipe` [386]

  Enable support for named pipes. This option applies only on Windows NT, 2000, XP, and 2003 systems, and can be used only with the `mysqld-nt` and `mysqld-max-nt` servers that support named-pipe connections.

- `--enable-pstack` [387]

  Print a symbolic stack trace on failure. This capability is available only on Intel Linux systems, and only if MySQL was configured with the `--with-pstack` option.

- `--exit-info[=flags]` [387], `-T [flags]`

  This is a bit mask of different flags that you can use for debugging the `mysqld` server. Do not use this option unless you know *exactly* what it does!

- `--external-locking` [387]

  Enable external locking (system locking), which is disabled by default as of MySQL 4.0. Note that if you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for `mysqld` to deadlock. This option was named `--enable-locking` before MySQL 4.0.3.

  For more information about external locking, including conditions under which it can and cannot be used, see Section 7.6.4, “External Locking”.

- `--flush` [387]

  Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.

- `--gdb` [387]

  Install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling. See Section 18.4, “Porting to Other Systems”. This option was added in MySQL 4.0.14.

- `--init-file=file_name` [387]

  Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

- `--innodb-safe-binlog` [387]

  If this option is given, then after a crash recovery by `InnoDB`, `mysqld` truncates the binary log after the last not-rolled-back transaction in the log. The option also causes `InnoDB` to print an error if the binary log is smaller or shorter than it should be. See Section 5.3.4, “The Binary Log”.

- `--innodb-xxx`

  Set an option for the `InnoDB` storage engine. The `InnoDB` options are listed in Section 13.2.4, “`InnoDB` Startup Options and System Variables”.

- `--install [service_name]` [387]

Command-Line Format	`--install [service_name]`

  (Windows only) Install the server as a Windows service that starts automatically during Windows startup. The default service name is `MySQL` if no `service_name` value is given. For more information, see Section 2.3.11, “Starting MySQL as a Windows Service”.

- `--install-manual [service_name]` [387]

Command-Line Format	`--install-manual [service_name]`

(Windows only) Install the server as a Windows service that must be started manually. It does not start automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.11, “Starting MySQL as a Windows Service”.

- `--language=`*`lang_name`*`, -L` *`lang_name`* [388]

  The language to use for error messages. *`lang_name`* can be given as the language name or as the full path name to the directory where the language files are installed. See Section 9.3, “Setting the Error Message Language”.

- `--log[=`*`file_name`*`]` [388], `-l [`*`file_name`*`]`

  Log connections and SQL statements received from clients to this file. See Section 5.3.2, “The General Query Log”. If you omit the file name, MySQL uses *`host_name`*`.log` as the file name.

- `--log-error[=`*`file_name`*`]` [388]

  Log errors and startup messages to this file. See Section 5.3.1, “The Error Log”. If you omit the file name, MySQL uses *`host_name`*`.err`. If the file name has no extension, the server adds an extension of `.err`.

- `--log-isam[=`*`file_name`*`]` [388]

  Log all `ISAM`/`MyISAM` changes to this file (used only when debugging `ISAM`/`MyISAM`).

- `--log-long-format` [388]

  Log extra information to the update log, binary update log, and slow query log, if they have been activated. For example, the user name and timestamp are logged for queries. Before MySQL 4.1, if you are using `--log-slow-queries` [388] and `--log-long-format` [388], queries that are not using indexes also are logged to the slow query log. `--log-long-format` [388] is deprecated as of MySQL version 4.1, when `--log-short-format` [388] was introduced. (Long log format is the default setting since version 4.1.) Also note that starting with MySQL 4.1, the `--log-queries-not-using-indexes` [388] option is available for the purpose of logging queries that do not use indexes to the slow query log.

- `--log-queries-not-using-indexes` [388]

  If you are using this option with the slow query log enabled, queries that are expected to retrieve all rows are logged. See Section 5.3.5, “The Slow Query Log”. This option does not necessarily mean that no index is used. For example, a query that uses a full index scan uses an index but would be logged because the index would not limit the number of rows. This option is available as of MySQL 4.1.

- `--log-short-format` [388]

  Originally intended to log less information to the update log, binary log and slow query log, if they have been activated. This option was introduced in MySQL 4.1, but is not operational.

- `--log-slow-admin-statements` [388]

  Log slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` to the slow query log.

  This option was added in MySQL 4.1.13. (It is unnecessary in MySQL 4.0 because slow administrative statements are logged by default.)

- `--log-slow-queries[=`*`file_name`*`]` [388]

Log all queries that have taken more than `long_query_time` [417] seconds to execute to this file. See Section 5.3.5, “The Slow Query Log”. Note that the default for the amount of information logged has changed in MySQL 4.1. See the `--log-long-format` [388] and `--log-short-format` [388] options for details.

- `--log-update[=file_name]` [389]

  Log updates to *`fileN`* where *`N`* is a unique number if not given. See Section 5.3.3, “The Update Log”. The update log is now deprecated; you should use the binary log instead (`--log-bin` [1181]). See Section 5.3.4, “The Binary Log”.

- `--log-warnings[=level]` [389], `-W [level]`

  Print out warnings such as `Aborted connection...` to the error log. Enabling this option by setting it greater than 0 is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). This option is enabled by default as of MySQL 4.0.19 and 4.1.2; to disable it, use `--log-warnings=0` [389]. As of MySQL 4.0.21 and 4.1.3, a *`level`* argument can be given. If omitted, the default *`level`* is 1. If the value is greater than 1, aborted connections are written to the error log. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

  If a slave server was started with `--log-warnings` [389] enabled, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth.

  Before MySQL 4.0.21 and 4.1.3, this is a boolean option, not an integer-valued option. Before 4.0, this option was named `--warnings` [389].

- `--low-priority-updates` [389]

  Give table-modifying operations (`INSERT`, `REPLACE`, `DELETE`, `UPDATE`) lower priority than selects. This can also be done using `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query, or by `SET LOW_PRIORITY_UPDATES=1` to change the priority in one thread. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`). See Section 7.6.2, “Table Locking Issues”.

- `--memlock` [389]

  Lock the `mysqld` process in memory. This option might help if you have a problem where the operating system is causing `mysqld` to swap to disk.

  `--memlock` [389] works on systems that support the `mlockall()` system call; this includes Solaris as well as most Linux distributions that use a 2.4 or newer kernel. On Linux systems, you can tell whether or not `mlockall()` (and thus this option) is supported by checking to see whether or not it is defined in the system `mman.h` file, like this:

```
shell> grep mlockall /usr/include/sys/mman.h
```

  If `mlockall()` is supported, you should see in the output of the previous command something like the following:

```
extern int mlockall (int __flags) __THROW;
```

Important

Using this option requires that you run the server as `root`, which, for reasons of security, is normally not a good idea. See Section 5.4.6, "How to Run MySQL as a Normal User".

You must not try to use this option on a system that does not support the `mlockall()` system call; if you do so, `mysqld` will very likely crash as soon as you try to start it.

- `--myisam-block-size=N` [390]

  The block size to be used for `MyISAM` index pages.

- `--myisam-recover[=option[,option]...]]` [390]

  Set the `MyISAM` storage engine recovery mode. The option value is any combination of the values of `DEFAULT`, `BACKUP`, `FORCE`, or `QUICK`. If you specify multiple values, separate them by commas. You can also use a value of `""` to disable this option. If this option is used, each time `mysqld` opens a `MyISAM` table, it checks whether the table is marked as crashed or was not closed properly. (The last option works only if you are running with external locking disabled.) If this is the case, `mysqld` runs a check on the table. If the table was corrupted, `mysqld` attempts to repair it.

  The following options affect how the repair works.

Option	Description
`DEFAULT`	Recovery without backup, forcing, or quick checking.
`BACKUP`	If the data file was changed during recovery, save a backup of the *`tbl_name`*`.MYD` file as *`tbl_name-datetime`*`.BAK`.
`FORCE`	Run recovery even if we would lose more than one row from the `.MYD` file.
`QUICK`	do not check the rows in the table if there are not any delete blocks.

  Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options `BACKUP,FORCE`. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

  See Section 13.1.1, "`MyISAM` Startup Options".

  This option is available as of MySQL 3.23.25.

- `--new`

  The `--new` option can be used to make the server behave as 4.1 in certain respects, easing a 4.0 to 4.1 upgrade:

  - Hexadecimal strings such as `0xFF` are treated as strings by default rather than as numbers. (Works in 4.0.12 and up.)
  - `TIMESTAMP` is returned as a string with the format `'YYYY-MM-DD HH:MM:SS'`. (Works in 4.0.13 and up.) See Chapter 10, *Data Types*.

  This option can be used to help you see how your applications behave in MySQL 4.1, without actually upgrading to 4.1.

- `--old-passwords` [391]

  Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See Section 5.4.2.3, “Password Hashing in MySQL”.

- `--old-protocol` [391], `-o`

  Use the 3.20 protocol for compatibility with some very old clients. This option was removed in MySQL 4.1.1.

- `--one-thread`

  Only use one thread (for debugging under Linux). This option is available only if the server is built with debugging enabled. See Section 18.4, “Porting to Other Systems”.

- `--open-files-limit=`*`count`* [391]

  Changes the number of file descriptors available to `mysqld`. You should try increasing the value of this option if `mysqld` gives you the error `Too many open files`. `mysqld` uses the option value to reserve descriptors with `setrlimit()`. If the requested number of file descriptors cannot be allocated, `mysqld` writes a warning to the error log.

  `mysqld` may attempt to allocate more than the requested number of descriptors (if they are available), using the values of `max_connections` [419] and `table_cache` [431] to estimate whether more descriptors will be needed.

- `--pid-file=`*`path`* [391]

  The path name of the process ID file. The server creates the file in the data directory unless an absolute path name is given to specify a different directory. This file is used by other programs such as `mysqld_safe` to determine the server's process ID.

- `--port=`*`port_num`* [391], `-P` *`port_num`*

  The port number to use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

- `--remove [`*`service_name`*`]` [391]

Command-Line Format	`--remove [service_name]`

  (Windows only) Remove a MySQL Windows service. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.11, “Starting MySQL as a Windows Service”.

- `--safe-mode` [391]

  Skip some optimization stages.

- `--safe-show-database` [391]

  With this option, the `SHOW DATABASES` statement displays only the names of those databases for which the user has some kind of privilege. As of MySQL 4.0.2, this option is deprecated and does not do anything (it is enabled by default), because there is a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See Section 5.5.1, “Privileges Provided by MySQL”.

- `--safe-user-create` [392]

  If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement, if the user does not have the `INSERT` [492] privilege for the `mysql.user` table or any column in the table.

- `--secure-auth` [392]

  Disallow authentication by clients that attempt to use accounts that have old (pre-4.1) passwords. This option is available as of MySQL 4.1.1.

- `--shared-memory` [392]

  Enable shared-memory connections by local clients. This option is available only on Windows. It was added in MySQL 4.1.0.

- `--shared-memory-base-name=`*name* [392]

  The name of shared memory to use for shared-memory connections. This option is available only on Windows. The default name is `MYSQL`. The name is case sensitive. This option was added in MySQL 4.1.0.

- `--skip-bdb` [392]

  Disable the `BDB` storage engine. This saves memory and might speed up some operations. Do not use this option if you require `BDB` tables.

- `--skip-concurrent-insert` [392]

  Turn off the ability to select and insert at the same time on `MyISAM` tables. (This is to be used only if you think you have found a bug in this feature.) See Section 7.6.3, “Concurrent Inserts”.

- `--skip-delay-key-write` [392]

  Ignore the `DELAY_KEY_WRITE` option for all tables. As of MySQL 4.0.3, you should use `--delay-key-write=OFF` [386] instead. See Section 7.8.2, “Tuning Server Parameters”.

- `--skip-external-locking` [392]

  Do not use external locking (system locking). For more information about external locking, including conditions under which it can and cannot be used, see Section 7.6.4, “External Locking”.

  External locking has been disabled by default since MySQL 4.0.

- `--skip-grant-tables` [392]

  This option causes the server to start without using the privilege system at all, which gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement after connecting to the server. This option also suppresses loading of user-defined functions (UDFs).

- `--skip-host-cache` [392]

  Do not use the internal host name cache for faster name-to-IP resolution. Instead, query the DNS server every time a client connects. See Section 7.8.5, “How MySQL Uses DNS”.

- `--skip-innodb` [1066]

Disable the `InnoDB` storage engine. In this case, the server will not start if the default storage engine is set to `InnoDB`. Use `--default-storage-engine` [386] to set the default to some other engine if necessary.

- `--skip-isam` [393]

  Disable the `ISAM` storage engine. As of MySQL 4.1, `ISAM` is disabled by default, so this option applies only if the server was configured with support for `ISAM`. This option was added in MySQL 4.1.1.

- `--skip-merge` [393]

  Disable the `MERGE` storage engine. This option was added in MySQL 4.1.21. It can be used if the following behavior is undesirable: If a user has access to `MyISAM` table *`t`*, that user can create a `MERGE` table *`m`* that accesses *`t`*. However, if the user's privileges on *`t`* are subsequently revoked, the user can continue to access *`t`* by doing so through *`m`*.

- `--skip-name-resolve` [393]

  Do not resolve host names when checking client connections. Use only IP addresses. If you use this option, all `Host` column values in the grant tables must be IP addresses or `localhost`. See Section 7.8.5, “How MySQL Uses DNS”.

- `--skip-networking` [393]

  Do not listen for TCP/IP connections at all. All interaction with `mysqld` must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. See Section 7.8.5, “How MySQL Uses DNS”.

- `--skip-new`

  Do not use new, possibly wrong routines.

- `--skip-symlink` [393]

  This is the old form of `--skip-symbolic-links` [393], for use before MySQL 4.0.13. `--skip-symlink` [393] is deprecated as of 4.0.13 and is removed in MySQL 5.5.

- `--ssl*`

  Options that begin with `--ssl` [521] specify whether to permit clients to connect using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, “SSL Command Options”.

- `--standalone` [393]

  Available on Windows NT-based systems only; instructs the MySQL server not to run as a service.

- `--symbolic-links` [393], `--skip-symbolic-links` [393]

  Enable or disable symbolic link support. This option has different effects on Windows and Unix:

  - On Windows, enabling symbolic links enables you to establish a symbolic link to a database directory by creating a *`db_name`*`.sym` file that contains the path to the real directory. See Section 7.10.3, “Using Symbolic Links for Databases on Windows”.

  - On Unix, enabling symbolic links means that you can link a `MyISAM` index file or data file to another directory with the `INDEX DIRECTORY` or `DATA DIRECTORY` options of the `CREATE TABLE` statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See Section 7.10.2, “Using Symbolic Links for Tables on Unix”.

This option was added in MySQL 4.0.13.

- `--skip-safemalloc` [394]

  If MySQL is configured with `--with-debug=full` [98], all MySQL programs check for memory overruns during each memory allocation and memory freeing operation. This checking is very slow, so for the server you can avoid it when you do not need it by using the `--skip-safemalloc` [394] option.

- `--skip-show-database` [394]

  This option sets the `skip_show_database` [427] system variable that controls who is permitted to use the `SHOW DATABASES` statement. See Section 5.1.3, “Server System Variables”.

- `--skip-stack-trace` [394]

  do not write stack traces. This option is useful when you are running `mysqld` under a debugger. On some systems, you also must use this option to get a core file. See Section 18.4, “Porting to Other Systems”.

- `--skip-thread-priority` [394]

  Disable using thread priorities for faster response time.

  `mysqld` makes a large number of invalid calls to thread scheduling routines on Linux. These calls do not affect performance noticeably but may be a source of “noise” for debugging tools. For example, they can overwhelm other information of more interest in kernel logs. To avoid these calls, start the server with the `--skip-thread-priority` [394] option.

- `--socket=`*`path`* [394]

  On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. If this option is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory. On Windows, the option specifies the pipe name to use when listening for local connections that use a named pipe. The default value is `MySQL` (not case sensitive).

- `--sql-mode=`*`value`*`[,`*`value`*`[,`*`value`*`...]]` [394]

  Set the SQL mode. See Section 5.1.6, “Server SQL Modes”. This option was added in 3.23.41.

- `--temp-pool` [394]

  This option causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to “leak” memory, because it is being allocated to the directory entry cache rather than to the disk cache.

- `--transaction-isolation=`*`level`* [394]

  Sets the default transaction isolation level. The `level` value can be `READ-UNCOMMITTED` [975], `READ-COMMITTED` [975], `REPEATABLE-READ` [976], or `SERIALIZABLE` [976]. See Section 12.3.6, “`SET TRANSACTION` Syntax”.

- `--tmpdir=`*`path`* [394], `-t` *`path`*

The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. Starting from MySQL 4.1.0, this option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (“`:`”) on Unix and semicolon characters (“`;`”) on Windows, NetWare, and OS/2. If the MySQL server is acting as a replication slave, you should not set `--tmpdir` [394] to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. For more information about the storage location of temporary files, see Section B.5.4.4, “Where MySQL Stores Temporary Files”. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

- `--user={user_name|user_id}` [395], `-u {user_name|user_id}`

  Run the `mysqld` server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

  This option is *mandatory* when starting `mysqld` as `root`. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as `root`. See Section 5.4.1, “General Security Guidelines”.

  Starting from MySQL 3.23.56 and 4.0.12: To avoid a possible security hole where a user adds a `--user=root` [395] option to a `my.cnf` file (thus causing the server to run as `root`), `mysqld` uses only the first `--user` [395] option specified and produces a warning if there are multiple `--user` [395] options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` [395] option in `/etc/my.cnf` and specify a value other than `root`. The option in `/etc/my.cnf` is found before any other `--user` [395] options, which ensures that the server runs as a user other than `root`, and that a warning results if any other `--user` [395] option is found.

- `--verbose` [395], `-v` [395]

  As of MySQL 4.1.1, use this option with the `--help` [384] option for detailed help.

- `--version` [395], `-V`

  Display version information and exit.

As of MySQL 4.0, you can assign a value to a server system variable by using an option of the form `--var_name=value`. For example, `--key_buffer_size=32M` [415] sets the `key_buffer_size` [415] variable to a value of 32MB.

Note that when you assign a value to a variable, MySQL might automatically correct the value to stay within a given range, or adjust the value to the closest permissible value if only certain values are permitted.

If you want to restrict the maximum value to which a variable can be set at runtime with `SET`, you can define this by using the `--maximum-var_name=value` command-line option.

It is also possible to set variables by using `--set-variable=var_name=value` or `--var_name=value` syntax. *This syntax is deprecated as of MySQL 4.0.*

You can change the values of most system variables for a running server with the `SET` statement. See Section 12.4.4, “`SET` Syntax”.

Section 5.1.3, “Server System Variables”, provides a full description for all variables, and additional information for setting them at server startup and runtime. Section 7.8.2, “Tuning Server Parameters”, includes information on optimizing the server by tuning system variables.

5.1.3 Server System Variables

The MySQL server maintains many system variables that indicate how it is configured. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. As of MySQL 4.0.3, most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

There are several ways to see the names and values of system variables:

- To see the values that a server will use based on its compiled-in defaults and any option files that it reads, use this command (omit `--verbose` [395] before MySQL 4.1.1):

```
mysqld --verbose --help
```

- To see the values that a server will use based on its compiled-in defaults, ignoring the settings in any option files, use this command (omit `--verbose` [395] before MySQL 4.1.1):

```
mysqld --no-defaults --verbose --help
```

  For more information, see Section 18.4.3, “The DBUG Package”.

- To see the current values used by a running server, use the `SHOW VARIABLES` statement.

This section provides a description of each system variable. Variables with no version indicated have been present since at least MySQL 3.22.

The following table lists all available system variables.

Table 5.2 System Variable Summary

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
autocommit [406]	Yes	Yes	Yes	Session	Yes
back_log [406]			Yes	Global	No
basedir [384]	Yes	Yes	Yes	Global	No
bdb_cache_size [406]			Yes	Global	No
bdb-home [1138]	Yes	Yes			No
- *Variable*: bdb_home [407]			Yes	Global	No
bdb-lock-detect [1138]	Yes	Yes	Yes	Global	No
bdb_log_buffer_size [407]			Yes	Global	No
bdb-logdir [1138]	Yes	Yes			No
- *Variable*: bdb_logdir [407]			Yes	Global	No
bdb_max_lock [407]			Yes	Global	No
bdb-shared-data [1138]	Yes	Yes			No
- *Variable*: bdb_shared_data [407]			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
bdb-tmpdir [1138]	Yes	Yes			No
- *Variable*: bdb_tmpdir [407]			Yes	Global	No
big-tables [384]	Yes	Yes			Yes
- *Variable*: big_tables [407]			Yes	Session	Yes
bind-address [384]	Yes	Yes	Yes	Global	No
binlog_cache_size [407]	Yes	Yes	Yes	Global	Yes
bulk_insert_buffer_size [408]	Yes	Yes	Yes	Both	Yes
character_set [408]	Yes		Yes	Global	No
character_set_client [408]			Yes	Both	Yes
character_set_connection [408]			Yes	Both	Yes
character_set_database [408][a]			Yes	Both	Yes
character_set_results [408]			Yes	Both	Yes
character-set-server [385]	Yes	Yes			Yes
- *Variable*: character_set_server [408]			Yes	Both	Yes
character_set_system [409]			Yes	Global	No
character-sets-dir [385]	Yes	Yes			No
- *Variable*: character_sets_dir [409]			Yes	Global	No
collation_connection [409]			Yes	Both	Yes
collation_database [409][b]			Yes	Both	Yes
collation-server [385]	Yes	Yes			Yes
- *Variable*: collation_server [409]			Yes	Both	Yes
concurrent_insert [409]	Yes	Yes	Yes	Global	Yes
connect_timeout [409]	Yes	Yes	Yes	Global	Yes
datadir [385]	Yes	Yes	Yes	Global	No
date_format [409]			Yes	Both	No
datetime_format [410]			Yes	Both	No
debug [385]	Yes	Yes	Yes	Both	Yes
default-storage-engine [386]	Yes	Yes	Yes	Both	Yes
default_week_format [410]	Yes	Yes	Yes	Both	Yes
delay-key-write [386]	Yes	Yes			Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- *Variable*: delay_key_write [410]			Yes	Global	Yes
delayed_insert_limit [410]	Yes	Yes	Yes	Global	Yes
delayed_insert_timeout [410]	Yes	Yes	Yes	Global	Yes
delayed_queue_size [410]	Yes	Yes	Yes	Global	Yes
error_count [410]			Yes	Session	No
expire_logs_days [411]	Yes	Yes	Yes	Global	Yes
flush [411]	Yes	Yes	Yes	Global	Yes
flush_time [411]	Yes	Yes	Yes	Global	Yes
foreign_key_checks [411]			Yes	Session	Yes
ft_boolean_syntax [411]	Yes	Yes	Yes	Global	Yes
ft_max_word_len [412]	Yes	Yes	Yes	Global	No
ft_min_word_len [412]	Yes	Yes	Yes	Global	No
ft_query_expansion_limit [412]	Yes	Yes	Yes	Global	No
ft_stopword_file [412]	Yes	Yes	Yes	Global	No
group_concat_max_len [412]	Yes	Yes	Yes	Both	Yes
have_archive [412]			Yes	Global	No
have_bdb [412]			Yes	Global	No
have_blackhole_engine [413]			Yes	Global	No
have_compress [413]			Yes	Global	No
have_crypt [413]			Yes	Global	No
have_csv [413]			Yes	Global	No
have_example_engine [413]			Yes	Global	No
have_geometry [413]			Yes	Global	No
have_innodb [413]			Yes	Global	No
have_isam [413]			Yes	Global	No
have_merge_engine [413]			Yes	Global	No
have_ndbcluster [1302]			Yes	Global	No
have_openssl [413]			Yes	Global	No
have_query_cache [413]			Yes	Global	No
have_raid [413]			Yes	Global	No
have_rtree_keys [413]			Yes	Global	No
have_symlink [413]			Yes	Global	No
identity [414]			Yes	Session	Yes
init_connect [414]	Yes	Yes	Yes	Global	Yes
init-file [387]	Yes	Yes			No
- *Variable*: init_file [414]			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
init_slave [1180]	Yes	Yes	Yes	Global	Yes
innodb_additional_mem_pool_size [1066]	Yes	Yes	Yes	Global	No
innodb_autoextend_increment [1067]	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_awe_mem_mb [1067]	Yes	Yes	Yes	Global	No
innodb_buffer_pool_size [1067]	Yes	Yes	Yes	Global	No
innodb_data_file_path [1067]	Yes	Yes	Yes	Global	No
innodb_data_home_dir [1068]	Yes	Yes	Yes	Global	No
innodb_fast_shutdown [1068]	Yes	Yes	Yes	Global	Yes
innodb_file_io_threads [1068]	Yes	Yes	Yes	Global	No
innodb_file_per_table [1068]	Yes	Yes	Yes	Global	No
innodb_flush_log_at_trx_commit [1068]	Yes	Yes	Yes	Global	Yes
innodb_flush_method [1069]	Yes	Yes	Yes	Global	No
innodb_force_recovery [1069]	Yes	Yes	Yes	Global	No
innodb_lock_wait_timeout [1069]	Yes	Yes	Yes	Global	No
innodb_locks_unsafe_for_binlog [1070]	Yes	Yes	Yes	Global	No
innodb_log_arch_dir [1071]	Yes	Yes	Yes	Global	No
innodb_log_archive [1071]	Yes	Yes	Yes	Global	No
innodb_log_buffer_size [1071]	Yes	Yes	Yes	Global	No
innodb_log_file_size [1071]	Yes	Yes	Yes	Global	No
innodb_log_files_in_group [1071]	Yes	Yes	Yes	Global	No
innodb_log_group_home_dir [1071]	Yes	Yes	Yes	Global	No
innodb_max_dirty_pages_pct [1071]	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag [1071]	Yes	Yes	Yes	Global	Yes
innodb_mirrored_log_groups [1072]	Yes	Yes	Yes	Global	No
innodb_open_files [1072]	Yes	Yes	Yes	Global	No
innodb_table_locks [1072]	Yes	Yes	Yes	Both	Yes
innodb_thread_concurrency [1072]	Yes	Yes	Yes	Global	Yes
insert_id [414]			Yes	Session	Yes
interactive_timeout [414]	Yes	Yes	Yes	Both	Yes
join_buffer_size [415]	Yes	Yes	Yes	Both	Yes
key_buffer_size [415]	Yes	Yes	Yes	Global	Yes
key_cache_age_threshold [416]	Yes	Yes	Yes	Global	Yes
key_cache_block_size [416]	Yes	Yes	Yes	Global	Yes
key_cache_division_limit [416]	Yes	Yes	Yes	Global	Yes
language [388]	Yes	Yes	Yes	Global	No
last_insert_id [416]			Yes	Session	Yes
lc_time_names [417]			Yes	Both	Yes
license [417]			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
local_infile [417]			Yes	Global	Yes
locked_in_memory [417]			Yes	Global	No
log [388]	Yes	Yes	Yes	Global	No
log_bin [1183]			Yes	Global	No
log-bin [1181]	Yes	Yes	Yes	Global	No
log-error [388]	Yes	Yes			No
- *Variable*: log_error [417]			Yes	Global	No
log-queries-not-using-indexes [388]	Yes	Yes			Yes
- *Variable*: log_queries_not_using_indexes			Yes	Global	Yes
log-slave-updates [1173]	Yes	Yes			No
- *Variable*: log_slave_updates [1183]			Yes	Global	No
log_slave_updates [1183]	Yes	Yes	Yes	Global	No
log-slow-queries [388]	Yes	Yes			No
- *Variable*: log_slow_queries [417]			Yes	Global	No
log-warnings [389]	Yes	Yes			Yes
- *Variable*: log_warnings [417]			Yes	Both	Yes
long_query_time [417]	Yes	Yes	Yes	Both	Yes
low-priority-updates [389]	Yes	Yes			Yes
- *Variable*: low_priority_updates [418]			Yes	Both	Yes
lower_case_file_system [418]			Yes	Global	No
lower_case_table_names [418]	Yes	Yes	Yes	Global	No
max_allowed_packet [418]	Yes	Yes	Yes	Global	Yes
max_binlog_cache_size [1183]	Yes	Yes	Yes	Global	Yes
max_binlog_size [1184]	Yes	Yes	Yes	Global	Yes
max_connect_errors [419]	Yes	Yes	Yes	Global	Yes
max_connections [419]	Yes	Yes	Yes	Global	Yes
max_delayed_threads [419]	Yes	Yes	Yes	Both	Yes
max_error_count [419]	Yes	Yes	Yes	Both	Yes
max_heap_table_size [419]	Yes	Yes	Yes	Both	Yes
max_insert_delayed_threads [419]			Yes	Both	Yes
max_join_size [419]	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
max_length_for_sort_data [420]	Yes	Yes	Yes	Both	Yes
max_prepared_stmt_count [420]	Yes	Yes	Yes	Global	Yes
max_relay_log_size [420]	Yes	Yes	Yes	Global	Yes
max_seeks_for_key [420]	Yes	Yes	Yes	Both	Yes
max_sort_length [420]	Yes	Yes	Yes	Both	Yes
max_user_connections [420]	Yes	Yes	Yes	Global	Yes
max_write_lock_count [421]	Yes	Yes	Yes	Global	Yes
memlock [389]	Yes	Yes	Yes	Global	No
myisam_data_pointer_size [421]	Yes	Yes	Yes	Global	Yes
myisam_max_extra_sort_file_size [421]	Yes	Yes	Yes	Global	No
myisam_max_sort_file_size [421]	Yes	Yes	Yes	Global	Yes
myisam_recover_options [421]			Yes	Global	No
myisam_repair_threads [421]	Yes	Yes	Yes	Both	Yes
myisam_sort_buffer_size [421]	Yes	Yes	Yes	Both	Yes
myisam_stats_method [422]	Yes	Yes	Yes	Both	Yes
named_pipe [422]			Yes	Global	No
ndb_autoincrement_prefetch_sz [1302]	Yes	Yes	Yes	Both	Yes
ndb_cache_check_time [1302]	Yes	Yes	Yes	Global	Yes
ndb_force_send [1303]	Yes	Yes	Yes	Both	Yes
ndb_use_exact_count [1305]			Yes	Both	Yes
ndb_use_transactions [1305]	Yes	Yes	Yes	Both	Yes
net_buffer_length [422]	Yes	Yes	Yes	Both	Yes
net_read_timeout [422]	Yes	Yes	Yes	Both	Yes
net_retry_count [422]	Yes	Yes	Yes	Both	Yes
net_write_timeout [422]	Yes	Yes	Yes	Both	Yes
new [390]	Yes	Yes	Yes	Both	Yes
old_passwords [423]			Yes	Both	Yes
open-files-limit [391]	Yes	Yes			No
- *Variable*: open_files_limit [423]			Yes	Global	No
pid-file [391]	Yes	Yes			No
- *Variable*: pid_file [423]			Yes	Global	No
plugin_dir [423]	Yes	Yes	Yes	Global	No
port [391]	Yes	Yes	Yes	Global	No
preload_buffer_size [423]	Yes	Yes	Yes	Both	Yes
prepared_stmt_count [423]			Yes	Both	No
protocol_version [423]			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
pseudo_thread_id [424]			Yes	Session	Yes
query_alloc_block_size [424]	Yes	Yes	Yes	Both	Yes
query_cache_limit [424]	Yes	Yes	Yes	Global	Yes
query_cache_min_res_unit [424]	Yes	Yes	Yes	Global	Yes
query_cache_size [424]	Yes	Yes	Yes	Global	Yes
query_cache_type [424]	Yes	Yes	Yes	Both	Yes
query_cache_wlock_invalidate [425]	Yes	Yes	Yes	Both	Yes
query_prealloc_size [425]	Yes	Yes	Yes	Both	Yes
rand_seed1 [425]			Yes	Session	Yes
rand_seed2 [425]			Yes	Session	Yes
range_alloc_block_size [425]	Yes	Yes	Yes	Both	Yes
read_buffer_size [425]	Yes	Yes	Yes	Both	Yes
read_only [425]	Yes	Yes	Yes	Global	Yes
read_rnd_buffer_size [426]	Yes	Yes	Yes	Both	Yes
relay-log [1175]	Yes	Yes			No
- *Variable*: relay_log			Yes	Global	No
relay-log-index [1175]	Yes	Yes			No
- *Variable*: relay_log_index			Yes	Global	No
relay_log_index	Yes	Yes	Yes	Global	No
relay_log_info_file	Yes	Yes	Yes	Global	No
relay_log_purge [426]	Yes	Yes	Yes	Global	Yes
relay_log_space_limit [426]	Yes	Yes	Yes	Global	No
report-host [1178]	Yes	Yes			No
- *Variable*: report_host			Yes	Global	No
report-password [1178]	Yes	Yes			No
- *Variable*: report_password			Yes	Global	No
report-port [1179]	Yes	Yes			No
- *Variable*: report_port			Yes	Global	No
report-user [1179]	Yes	Yes			No
- *Variable*: report_user			Yes	Global	No
rpl_recovery_rank [1180]			Yes	Global	Yes
safe-show-database [391]	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
secure-auth [392]	Yes	Yes			Yes
- *Variable*: secure_auth [426]			Yes	Global	Yes
server-id [1167]	Yes	Yes			Yes
- *Variable*: server_id [426]			Yes	Global	Yes
shared_memory [426]			Yes	Global	No
shared_memory_base_name [427]			Yes	Global	No
skip_external_locking [427]	Yes	Yes	Yes	Global	No
skip-name-resolve [393]	Yes	Yes			No
- *Variable*: skip_name_resolve			Yes	Global	No
skip-networking [393]	Yes	Yes			No
- *Variable*: skip_networking [427]			Yes	Global	No
skip-show-database [394]	Yes	Yes			No
- *Variable*: skip_show_database [427]			Yes	Global	No
skip-sync-bdb-logs [1138]	Yes	Yes	Yes	Global	No
slave_compressed_protocol [1180]	Yes	Yes	Yes	Global	Yes
slave-load-tmpdir [1179]	Yes	Yes			No
- *Variable*: slave_load_tmpdir [1180]			Yes	Global	No
slave-net-timeout [1179]	Yes	Yes			Yes
- *Variable*: slave_net_timeout [1181]			Yes	Global	Yes
slave-skip-errors [1179]	Yes	Yes			No
- *Variable*: slave_skip_errors [1181]			Yes	Global	No
slave_transaction_retries [1181]	Yes	Yes	Yes	Global	Yes
slow_launch_time [427]	Yes	Yes	Yes	Global	Yes
socket [394]	Yes	Yes	Yes	Global	No
sort_buffer_size [427]	Yes	Yes	Yes	Both	Yes
sql_auto_is_null [428]			Yes	Session	Yes
sql_big_selects [428]			Yes	Session	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
sql_big_tables [407]			Yes	Session	Yes
sql_buffer_result [428]			Yes	Session	Yes
sql_log_bin [429]			Yes	Session	Yes
sql_log_off [429]			Yes	Session	Yes
sql_log_update [429]			Yes	Session	Yes
sql_low_priority_updates [418]			Yes	Both	Yes
sql_max_join_size [419]			Yes	Both	Yes
sql-mode [394]	Yes	Yes			Yes
- *Variable*: sql_mode [429]			Yes	Both	Yes
sql_notes [429]			Yes	Session	Yes
sql_quote_show_create [430]			Yes	Session	Yes
sql_safe_updates [430]			Yes	Session	Yes
sql_select_limit [430]			Yes	Both	Yes
sql_slave_skip_counter [1181]			Yes	Global	Yes
sql_warnings [430]			Yes	Session	Yes
ssl-ca [522]	Yes	Yes			No
- *Variable*: ssl_ca			Yes	Global	No
ssl-capath [522]	Yes	Yes			No
- *Variable*: ssl_capath			Yes	Global	No
ssl-cert [522]	Yes	Yes			No
- *Variable*: ssl_cert			Yes	Global	No
ssl-cipher [522]	Yes	Yes			No
- *Variable*: ssl_cipher			Yes	Global	No
ssl-key [522]	Yes	Yes			No
- *Variable*: ssl_key			Yes	Global	No
storage_engine [430]			Yes	Both	Yes
sync-bdb-logs [1138]	Yes	Yes	Yes	Global	No
sync_binlog [1184]	Yes	Yes	Yes	Global	Yes
sync_frm [430]	Yes	Yes	Yes	Global	Yes
system_time_zone [430]			Yes	Global	No
table_cache [431]	Yes	Yes	Yes	Global	Yes
table_type [430]			Yes	Both	Yes
thread_cache_size [431]	Yes	Yes	Yes	Global	Yes
thread_concurrency [431]	Yes	Yes	Yes	Global	No
thread_stack [431]	Yes	Yes	Yes	Global	No
time_format [431]			Yes	Both	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
time_zone [431]			Yes	Both	Yes
timestamp [432]			Yes	Session	Yes
tmp_table_size [432]	Yes	Yes	Yes	Both	Yes
tmpdir [394]	Yes	Yes	Yes	Global	No
transaction_alloc_block_size [432]	Yes	Yes	Yes	Both	Yes
transaction_prealloc_size [432]	Yes	Yes	Yes	Both	Yes
tx_isolation [433]			Yes	Both	Yes
unique_checks [433]			Yes	Session	Yes
version [433]			Yes	Global	No
version_comment [433]			Yes	Global	No
version_compile_machine [434]			Yes	Global	No
version_compile_os [434]			Yes	Global	No
wait_timeout [434]	Yes	Yes	Yes	Both	Yes
warning_count [434]			Yes	Session	No

[a] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

[b] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

For additional system variable information, see these sections:

- Section 5.1.4, “Using System Variables”, discusses the syntax for setting and displaying system variable values.
- Section 5.1.4.2, “Dynamic System Variables”, lists the variables that can be set at runtime.
- Information on tuning system variables can be found in Section 7.8.2, “Tuning Server Parameters”.
- Section 13.2.4, “`InnoDB` Startup Options and System Variables”, lists `InnoDB` system variables.
- Section 15.3.4.3, “MySQL Cluster System Variables”, lists system variables which are specific to MySQL Cluster.
- For information on server system variables specific to replication, see Section 14.8, “Replication and Binary Logging Options and Variables”.

Note

Some of the following variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` [410] works but `--delay_key_write=ON` [410] does not.

Some system variables control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to a system variable that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also

possible that the server will adjust a value upward. For example, if you assign a value of 0 to a variable for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some system variables take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued variable is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- `ansi_mode`

  This is `ON` if `mysqld` was started with `--ansi` [384]. See Section 1.9.3, “Running MySQL in ANSI Mode”. This variable was added in MySQL 3.23.6 and removed in 3.23.41. See the description for `sql_mode` [429].

  Depending on the network configuration of your system and the `Host` values for your accounts, clients may need to connect using an explicit `--host` option, such as `--host=localhost` or `--host=127.0.0.1`.

- `autocommit` [406]

  The autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0, you must use `COMMIT` to accept a transaction or `ROLLBACK` to cancel it. If `autocommit` [406] is 0 and you change it to 1, MySQL performs an automatic `COMMIT` of any open transaction. Another way to begin a transaction is to use a `START TRANSACTION` or `BEGIN` statement. See Section 12.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”.

  By default, client connections begin with `autocommit` [406] set to 1. To cause clients to begin with a default of 0, set the server's `init_connect` [414] system variable. See the description of that variable for instructions that show how to do this.

- `back_log` [406]

  The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The `back_log` [406] value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

  In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix `listen()` system call should have more details. Check your OS documentation for the maximum value for this variable. `back_log` [406] cannot be set higher than your operating system limit.

- `basedir` [406]

  The MySQL installation base directory. This variable can be set with the `--basedir` [384] option. Relative path names for other variables usually are resolved relative to the base directory.

- `bdb_cache_size` [406]

  The size of the buffer that is allocated for caching indexes and rows for `BDB` tables. If you do not use `BDB` tables, you should start `mysqld` with `--skip-bdb` [392] to not allocate memory for this cache. This variable was added in MySQL 3.23.14.

- `bdb_home` [407]

  The base directory for `BDB` tables. This should be assigned the same value as the `datadir` [409] variable. This variable was added in MySQL 3.23.14.

- `bdb_log_buffer_size` [407]

  The size of the buffer that is allocated for caching indexes and rows for `BDB` tables. If you do not use `BDB` tables, you should set this to 0 or start `mysqld` with `--skip-bdb` [392] in order not to allocate memory for this cache. This variable was added in MySQL 3.23.31.

- `bdb_logdir` [407]

  The directory where the `BDB` storage engine writes its log files. This variable can be set with the `--bdb-logdir` [1138] option. This variable was added in MySQL 3.23.14.

- `bdb_max_lock` [407]

  The maximum number of locks that can be active for a `BDB` table (10,000 by default). You should increase this value if errors such as the following occur when you perform long transactions or when `mysqld` has to examine many rows to calculate a query:

  ```
  bdb: Lock table is out of available locks
  Got error 12 from ...
  ```

  This variable was added in MySQL 3.23.29.

- `bdb_shared_data` [407]

  This is `ON` if you are using `--bdb-shared-data` [1138] to start Berkeley DB in multi-process mode. (Do not use `DB_PRIVATE` when initializing Berkeley DB.) This variable was added in MySQL 3.23.29.

- `bdb_tmpdir` [407]

  The `BDB` temporary file directory. This variable was added in MySQL 3.23.14.

- `bdb_version` [407]

  See the description for `version_bdb` [433].

- `big_tables` [407]

  If set to 1, all temporary tables are stored on disk rather than in memory. This is a little slower, but the error `The table tbl_name is full` does not occur for `SELECT` operations that require a large temporary table. The default value for a new connection is 0 (use in-memory temporary tables). As of MySQL 4.0, you should normally never need to set this variable, because MySQL automatically converts in-memory tables to disk-based tables as necessary.

  Note

  This variable was formerly named `sql_big_tables`.

- `binlog_cache_size` [407]

  The size of the cache to hold the SQL statements for the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and, starting from MySQL 4.1.2, if the server has the binary log enabled (`--log-bin` [1181] option). If you often use

large, multiple-statement transactions, you can increase this cache size to get more performance. The `Binlog_cache_use` [449] and `Binlog_cache_disk_use` [449] status variables can be useful for tuning the size of this variable. This variable was added in MySQL 3.23.29. See Section 5.3.4, "The Binary Log".

- `bulk_insert_buffer_size` [408]

  `MyISAM` uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA INFILE` when adding data to nonempty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB. Before MySQL 4.0.3. this variable was named `myisam_bulk_insert_tree_size`.

- `character_set` [408]

  The default character set. This variable was added in MySQL 3.23.3, then removed in MySQL 4.1.1 and replaced by the various `character_set_`*`xxx`* variables.

- `character_set_client` [408]

  The character set for statements that arrive from the client. This variable was added in MySQL 4.1.1.

  The session value of this variable is set using the character set requested by the client when the client connects to the server. (Many clients support a `--default-character-set` option to enable this character set to be specified explicitly. See also Section 9.1.4, "Connection Character Sets and Collations".) The global value of the variable is used to set the session value in cases when the client-requested value is unknown or not available, or the server is configured to ignore client requests:

  - The client is from a version of MySQL older than MySQL 4.1, and thus does not request a character set.
  - The client requests a character set not known to the server. For example, a Japanese-enabled client requests `sjis` when connecting to a server not configured with `sjis` support.
  - `mysqld` was started with the `--skip-character-set-client-handshake` [385] option, which causes it to ignore client character set configuration. This reproduces MySQL 4.0 behavior and is useful should you wish to upgrade the server without upgrading all the clients.

  `ucs2` cannot be used as a client character set, which means that it also does not work for `SET NAMES` or `SET CHARACTER SET`.

- `character_set_connection` [408]

  The character set used for literals that do not have a character set introducer and for number-to-string conversion. This variable was added in MySQL 4.1.1.

- `character_set_database` [408]

  The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `character_set_server` [408]. This variable was added in MySQL 4.1.1.

- `character_set_results` [408]

  The character set used for returning query results such as result sets or error messages to the client. This variable was added in MySQL 4.1.1.

- `character_set_server` [408]

The server default character set. This variable was added in MySQL 4.1.1.

- `character_set_system` [409]

  The character set used by the server for storing identifiers. The value is always `utf8`. This variable was added in MySQL 4.1.1.

- `character_sets` [409]

  The supported character sets. This variable was added in MySQL 3.23.15 and removed in MySQL 4.1.1. (Use `SHOW CHARACTER SET` for a list of character sets.)

- `character_sets_dir` [409]

  The directory where character sets are installed. This variable was added in MySQL 4.1.2.

- `collation_connection` [409]

  The collation of the connection character set. This variable was added in MySQL 4.1.1.

- `collation_database` [409]

  The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `collation_server` [409]. This variable was added in MySQL 4.1.1.

- `collation_server` [409]

  The server default collation. This variable was added in MySQL 4.1.1.

- `concurrent_insert` [409]

  If `ON` (the default), MySQL permits `INSERT` and `SELECT` statements to run concurrently for `MyISAM` tables that have no free blocks in the middle of the data file. If `OFF`, concurrent inserts are disabled. If you start `mysqld` with `--skip-new` [390], this variable is set to `OFF`. This variable was added in MySQL 3.23.7.

  See also Section 7.6.3, “Concurrent Inserts”.

- `connect_timeout` [409]

  The number of seconds that the `mysqld` server waits for a connect packet before responding with `Bad handshake`. The default value is 5 seconds.

  Increasing the `connect_timeout` [409] value might help if clients frequently encounter errors of the form `Lost connection to MySQL server at '`*`XXX`*`', system error: `*`errno`*.

- `convert_character_set` [409]

  The current character set mapping that was set by `SET CHARACTER SET`. This variable was removed in MySQL 4.1.

- `datadir` [409]

  The MySQL data directory. This variable can be set with the `--datadir` [385] option.

- `date_format` [409]

  This variable is unused.

- `datetime_format` [410]

  This variable is unused.

- `default_week_format` [410]

  The default mode value to use for the `WEEK()` [843] function. See Section 11.7, “Date and Time Functions”. This variable is available as of MySQL 4.0.14.

- `delay_key_write` [410]

  This option applies only to `MyISAM` tables. It can have one of the following values to affect handling of the `DELAY_KEY_WRITE` table option that can be used in `CREATE TABLE` statements.

Option	Description
`OFF`	`DELAY_KEY_WRITE` is ignored.
`ON`	MySQL honors any `DELAY_KEY_WRITE` option specified in `CREATE TABLE` statements. This is the default value.
`ALL`	All new opened tables are treated as if they were created with the `DELAY_KEY_WRITE` option enabled.

  If `DELAY_KEY_WRITE` is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all `MyISAM` tables by starting the server with the `--myisam-recover` [390] option (for example, `--myisam-recover=BACKUP,FORCE` [390]). See Section 5.1.2, “Server Command Options”, and Section 13.1.1, “`MyISAM` Startup Options”.

  Warning

  If you enable external locking with `--external-locking` [387], there is no protection against index corruption for tables that use delayed key writes.

  This variable was added in MySQL 3.23.8.

- `delayed_insert_limit` [410]

  After inserting `delayed_insert_limit` [410] delayed rows, the `INSERT DELAYED` handler thread checks whether there are any `SELECT` statements pending. If so, it permits them to execute before continuing to insert delayed rows.

- `delayed_insert_timeout` [410]

  How many seconds an `INSERT DELAYED` handler thread should wait for `INSERT` statements before terminating.

- `delayed_queue_size` [410]

  This is a per-table limit on the number of rows to queue when handling `INSERT DELAYED` statements. If the queue becomes full, any client that issues an `INSERT DELAYED` statement waits until there is room in the queue again.

- `error_count` [410]

  The number of errors that resulted from the last statement that generated messages. This variable is read only. See Section 12.4.5.11, “`SHOW ERRORS` Syntax”.

This variable was added in MySQL 4.1.0.

- `expire_logs_days` [411]

  The number of days for automatic binary log file removal. The default is 0, which means "no automatic removal." Possible removals happen at startup and when the binary log is flushed. Log flushing occurs as indicated in Section 5.3, "MySQL Server Logs". This variable was added in MySQL 4.1.0.

  To remove binary log files manually, use the `PURGE BINARY LOGS` statement. See Section 12.5.1.1, "`PURGE BINARY LOGS` Syntax".

- `flush` [411]

  If `ON`, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.5.4.2, "What to Do If MySQL Keeps Crashing". This variable is set to `ON` if you start `mysqld` with the `--flush` [387] option. This variable was added in MySQL 3.22.9.

- `flush_time` [411]

  If this is set to a nonzero value, all tables are closed every `flush_time` [411] seconds to free up resources and synchronize unflushed data to disk. This option is best used only on Windows 9x or Me, or on systems with minimal resources. This variable was added in MySQL 3.22.18.

- `foreign_key_checks` [411]

  If set to 1 (the default), foreign key constraints for `InnoDB` tables are checked. If set to 0, they are ignored. Disabling foreign key checking can be useful for reloading `InnoDB` tables in an order different from that required by their parent/child relationships. This variable was added in MySQL 3.23.52. See Section 13.2.5.4, "`FOREIGN KEY` Constraints".

  Setting `foreign_key_checks` [411] to 0 also affects data definition statements: `DROP DATABASE` drops a database even if it contains tables that have foreign keys that are referred to by tables outside the database, and `DROP TABLE` drops tables that have foreign keys that are referred to by other tables.

  Note

  Setting `foreign_key_checks` [411] to 1 does not trigger a scan of the existing table data. Therefore, rows added to the table while `foreign_key_checks = 0` [411] will not be verified for consistency.

- `ft_boolean_syntax` [411]

  The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. See Section 11.9.2, "Boolean Full-Text Searches". This variable was added as a read-only variable in MySQL 4.0.1. It can be modified as of MySQL 4.1.2.

  The default variable value is `'+ -><()~*:""&|'`. The rules for changing the value are as follows:

  - Operator function is determined by position within the string.
  - The replacement value must be 14 characters.
  - Each character must be an ASCII nonalphanumeric character.
  - Either the first or second character must be a space.

  - No duplicates are permitted except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
  - Positions 10, 13, and 14 (which by default are set to ":", "&", and "|") are reserved for future extensions.

- `ft_max_word_len` [412]

  The maximum length of the word to be included in a `FULLTEXT` index. This variable was added in MySQL 4.0.0.

  Note

  `FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len` [412]

  The minimum length of the word to be included in a `FULLTEXT` index. This variable was added in MySQL 4.0.0.

  Note

  `FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_query_expansion_limit` [412]

  The number of top matches to use for full-text searches performed using `WITH QUERY EXPANSION`. This variable was added in MySQL 4.1.1.

- `ft_stopword_file` [412]

  The file from which to read the list of stopwords for full-text searches. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the `myisam/ft_static.c` file). Setting this variable to the empty string (`''`) disables stopword filtering. See also Section 11.9.4, "Full-Text Stopwords". This variable was added in MySQL 4.0.10.

  Note

  `FULLTEXT` indexes must be rebuilt after changing this variable or the contents of the stopword file. Use `REPAIR TABLE tbl_name QUICK`.

- `group_concat_max_len` [412]

  The maximum permitted result length in bytes for the `GROUP_CONCAT()` [883] function. The default is 1024. This variable was added in MySQL 4.1.0.

- `have_archive` [412]

  `YES` if `mysqld` supports `ARCHIVE` tables, `NO` if not. This variable was added in MySQL 4.1.3.

- `have_bdb` [412]

  `YES` if `mysqld` supports `BDB` tables. `DISABLED` if `--skip-bdb` [392] is used. This variable was added in MySQL 3.23.30.

- have_blackhole_engine [413]

  YES if mysqld supports BLACKHOLE tables, NO if not. This variable was added in MySQL 4.1.11.

- have_compress [413]

  YES if the zlib compression library is available to the server, NO if not. If not, the COMPRESS() [866] and UNCOMPRESS() [869] functions cannot be used. This variable was added in MySQL 4.1.1.

- have_crypt [413]

  YES if the crypt() system call is available to the server, NO if not. If not, the ENCRYPT() [867] function cannot be used. This variable was added in MySQL 4.0.10.

- have_csv [413]

  YES if mysqld supports ARCHIVE tables, NO if not. This variable was added in MySQL 4.1.4.

- have_example_engine [413]

  YES if mysqld supports EXAMPLE tables, NO if not. This variable was added in MySQL 4.1.4.

- have_geometry [413]

  YES if the server supports spatial data types, NO if not. This variable was added in MySQL 4.1.3.

- have_innodb [413]

  YES if mysqld supports InnoDB tables. DISABLED if --skip-innodb [1066] is used. This variable was added in MySQL 3.23.37.

- have_isam [413]

  YES if mysqld supports ISAM tables. DISABLED if --skip-isam [393] is used. This variable was added in MySQL 3.23.30.

- have_merge_engine [413]

  YES if mysqld supports MERGE tables. DISABLED if --skip-merge [393] is used. This variable was added in MySQL 4.1.21.

- have_openssl [413]

  YES if mysqld supports SSL (encryption) connections, NO if not. This variable was added in MySQL 3.23.43.

- have_query_cache [413]

  YES if mysqld supports the query cache, NO if not. This variable was added in MySQL 4.0.2.

- have_raid [413]

  YES if mysqld supports the RAID option, NO if not. This variable was added in MySQL 3.23.30.

- have_rtree_keys [413]

  YES if RTREE indexes are available, NO if not. (These are used for spatial indexes in MyISAM tables.) This variable was added in MySQL 4.1.3.

- have_symlink [413]

YES if symbolic link support is enabled, NO if not. This is required on Unix for support of the DATA DIRECTORY and INDEX DIRECTORY table options, and on Windows for support of data directory symlinks.

This variable was added in MySQL 4.0.0.

- identity [414]

  This variable is a synonym for the last_insert_id [416] variable. It exists for compatibility with other database systems. As of MySQL 3.23.25, you can read its value with SELECT @@identity. As of MySQL 4.0.3, you can also set its value with SET identity.

- init_connect [414]

  A string to be executed by the server for each client that connects. The string consists of one or more SQL statements. To specify multiple statements, separate them by semicolon characters. For example, each client begins by default with autocommit mode enabled. There is no global system variable to specify that autocommit should be disabled by default, but init_connect [414] can be used to achieve the same effect:

  ```
  SET GLOBAL init_connect='SET autocommit=0';
  ```

  This variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

  ```
  [mysqld]
  init_connect='SET autocommit=0'
  ```

  The content of init_connect [414] is not executed for users that have the SUPER [493] privilege. This is done so that an erroneous value for init_connect [414] does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing init_connect [414] for users that have the SUPER [493] privilege enables them to open a connection and fix the init_connect [414] value.

  This variable was added in MySQL 4.1.2.

- init_file [414]

  The name of the file specified with the --init-file [387] option when you start the server. This should be a file containing SQL statements that you want the server to execute when it starts. Each statement must be on a single line and should not include comments. No statement terminator such as ;, \g, or \G should be given at the end of each statement. This variable was added in MySQL 3.23.2.

- innodb_*xxx*

  InnoDB system variables are listed in Section 13.2.4, “InnoDB Startup Options and System Variables”.

- insert_id [414]

  The value to be used by the following INSERT or ALTER TABLE statement when inserting an AUTO_INCREMENT value. This is mainly used with the binary log.

- interactive_timeout [414]

The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout` [434].

- `join_buffer_size` [415]

  The minimum size of the buffer that is used for plain index scans, range index scans, and joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` [415] to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary. There is no gain from setting the buffer larger than required to hold each matching row, and all joins allocate at least the minimum size, so use caution in setting this variable to a large value globally. It is better to keep the global setting small and change to a larger setting only in sessions that are doing large joins. Memory allocation time can cause substantial performance drops if the global size is larger than needed by most queries that use it.

  For additional information about join buffering, see Section 7.3.1.6, “Nested-Loop Join Algorithms”.

- `key_buffer_size` [415]

  Index blocks for `MyISAM` and `ISAM` tables are buffered and are shared by all threads. `key_buffer_size` [415] is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

  The maximum permissible setting for `key_buffer_size` [415] is 4GB. The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform. The value of this variable indicates the amount of memory requested. Internally, the server allocates as much memory as possible up to this amount, but the actual allocation might be less.

  You can increase the value to get better index handling for all reads and multiple writes; on a system whose primary function is to run MySQL using the `MyISAM` storage engine, 25% of the machine's total memory is an acceptable value for this variable. However, you should be aware that, if you make the value too large (for example, more than 50% of the machine's total memory), your system might start to page and become extremely slow. This is because MySQL relies on the operating system to perform file system caching for data reads, so you must leave some room for the file system cache. You should also consider the memory requirements of any other storage engines that you may be using in addition to `MyISAM`.

  For even more speed when writing many rows at the same time, use `LOCK TABLES`. See Section 7.3.2.1, “Speed of `INSERT` Statements”.

  You can check the performance of the key buffer by issuing a `SHOW STATUS` statement and examining the `Key_read_requests` [452], `Key_reads` [452], `Key_write_requests` [452], and `Key_writes` [452] status variables. (See Section 12.4.5, “`SHOW` Syntax”.) The `Key_reads/Key_read_requests` ratio should normally be less than 0.01. The `Key_writes/Key_write_requests` ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the `DELAY_KEY_WRITE` table option.

  The fraction of the key buffer in use can be determined using `key_buffer_size` [415] in conjunction with the `Key_blocks_unused` [452] status variable and the buffer block size. From MySQL 4.1.1 on, the buffer block size is available from the `key_cache_block_size` [416] server variable. The fraction of the buffer in use is:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer is allocated internally for administrative structures. Factors that influence the amount of overhead for these structures include block size and pointer size. As block size increases, the percentage of the key buffer lost to overhead tends to decrease. Larger blocks results in a smaller number of read operations (because more keys are obtained per read), but conversely an increase in reads of keys that are not examined (if not all keys in a block are relevant to a query).

Before MySQL 4.1.1, key cache blocks are 1024 bytes, and before MySQL 4.1.2, `Key_blocks_unused` [452] is unavailable. The `Key_blocks_used` [452] variable can be used as follows to determine the fraction of the key buffer in use:

```
(Key_blocks_used × 1024) / key_buffer_size
```

However, `Key_blocks_used` [452] indicates the maximum number of blocks that have ever been in use at once, so this formula does not necessarily represent the current fraction of the buffer that is in use.

As of MySQL 4.1, it is possible to create multiple `MyISAM` key caches. The size limit of 4GB applies to each cache individually, not as a group. See Section 7.5.1, “The `MyISAM` Key Cache”.

- `key_cache_age_threshold` [416]

  This value controls the demotion of buffers from the hot sublist of a key cache to the warm sublist. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. This variable was added in MySQL 4.1.1. See Section 7.5.1, “The `MyISAM` Key Cache”.

- `key_cache_block_size` [416]

  The size in bytes of blocks in the key cache. The default value is 1024. This variable was added in MySQL 4.1.1. See Section 7.5.1, “The `MyISAM` Key Cache”.

- `key_cache_division_limit` [416]

  The division point between the hot and warm sublists of the key cache buffer list. The value is the percentage of the buffer list to use for the warm sublist. Permissible values range from 1 to 100. The default value is 100. This variable was added in MySQL 4.1.1. See Section 7.5.1, “The `MyISAM` Key Cache”.

- `language` [416]

  The directory where error messages are located. See Section 9.3, “Setting the Error Message Language”.

- `large_files_support` [416]

  Whether `mysqld` was compiled with options for large file support. This variable was added in MySQL 3.23.28.

- `last_insert_id` [416]

  The value to be returned from `LAST_INSERT_ID()` [874]. This is stored in the binary log when you use `LAST_INSERT_ID()` [874] in a statement that updates a table. Setting this variable does not update the value returned by the `mysql_insert_id()` C API function.

- `lc_time_names` [417]

  This variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()` [832], `DAYNAME()` [833] and `MONTHNAME()` [837] functions. Locale names are POSIX-style values such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting. For further information, see Section 9.8, “MySQL Server Locale Support”. This variable was added in MySQL 4.1.21.

- `license` [417]

  The type of license the server has. This variable was added in MySQL 4.0.19.

- `local_infile` [417]

  Whether `LOCAL` is supported for `LOAD DATA INFILE` statements. See Section 5.4.5, “Security Issues with `LOAD DATA LOCAL`”. This variable was added in MySQL 4.0.3.

- `locked_in_memory` [417]

  Whether `mysqld` was locked in memory with `--memlock` [389]. This variable was added in MySQL 3.23.25.

- `log` [417]

  Whether logging of all statements to the general query log is enabled. See Section 5.3.2, “The General Query Log”.

- `log_error` [417]

  The location of the error log. This variable was added in MySQL 4.0.10.

- `log_slow_queries` [417]

  Whether slow queries should be logged. “Slow” is determined by the value of the `long_query_time` [417] variable. This variable was added in MySQL 4.0.2. See Section 5.3.5, “The Slow Query Log”.

- `log_update` [417]

  Whether the update log is enabled. This variable was added in MySQL 3.22.18. Note that the binary log is preferable to the update log, which is unavailable as of MySQL 5.0. See Section 5.3.3, “The Update Log”.

- `log_warnings` [417]

  Whether to produce additional warning messages. This variable was added in MySQL 4.0.3. It is enabled by default as of MySQL 4.0.19 and 4.1.2. As of MySQL 4.0.21 and 4.1.3, the variable can take values greater than 1 and aborted connections are not logged to the error log unless the value is greater than 1.

- `long_query_time` [417]

  If a query takes longer than this many seconds, the server increments the `Slow_queries` [455] status variable. If you are using the `--log-slow-queries` [388] option, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum value is 1. The default is 10. See Section 5.3.5, “The Slow Query Log”.

- `low_priority_updates` [418]

  If set to `1`, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`). Before MySQL 3.22.5, this variable was named `sql_low_priority_updates`.

- `lower_case_file_system` [418]

  This variable describes the case sensitivity of file names on the file system where the data directory is located. `OFF` means file names are case sensitive, `ON` means they are not case sensitive. This variable is read only because it reflects a file system attribute and setting it would have no effect on the file system. This variable was added in MySQL 4.0.19.

- `lower_case_table_names` [418]

  If set to 0, table names are stored as specified and comparisons are case sensitive. If set to 1, table names are stored in lowercase on disk and comparisons are not case sensitive. This variable was added in MySQL 3.23.6. If set to 2 (new in 4.0.18), table names are stored as given but compared in lowercase. From MySQL 4.0.2, this option also applies to database names. From 4.1.1, it also applies to table aliases. For additional information, see Section 8.2.2, “Identifier Case Sensitivity”.

  You should *not* set this variable to 0 if you are running MySQL on a system that does not have case-sensitive file names (such as Windows or Mac OS X). If you set this variable to 0 on such a system and access `MyISAM` tablenames using different lettercases, index corruption may result. *New in 4.0.18*: If this variable is not set at startup and the file system on which the data directory is located does not have case-sensitive file names, MySQL automatically sets `lower_case_table_names` [418] to 2.

  If you are using `InnoDB` tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

  The setting of this variable has no effect on replication filtering options. See Section 14.9, “How Servers Evaluate Replication Filtering Rules”, for more information.

  You should not use different settings for `lower_case_table_names` on replication masters and slaves. In particular, you should not do this when the slave uses a case-sensitive file system, as this can cause replication to fail. For more information, see Section 14.7.20, “Replication and Variables”.

- `max_allowed_packet` [418]

  The maximum size of one packet or any generated/intermediate string.

  The packet message buffer is initialized to `net_buffer_length` [422] bytes, but can grow up to `max_allowed_packet` [418] bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

  You must increase this value if you are using large `BLOB` columns or long strings. It should be as big as the largest `BLOB` you want to use. The protocol limit for `max_allowed_packet` [418] is 16MB before MySQL 4.0 and 1GB thereafter. The value should be a multiple of 1024; nonmultiples are rounded down to the nearest multiple.

  When you change the message buffer size by changing the value of the `max_allowed_packet` [418] variable, you should also change the buffer size on the client side if your client program permits it. The default `max_allowed_packet` [418] value built in to the client library is 1GB, but individual client programs might override this. For example, `mysql` and `mysqldump` have defaults

of 16MB and 24MB, respectively. They also enable you to change the client-side value by setting `max_allowed_packet` [418] on the command line or in an option file.

- `max_connect_errors` [419]

  If there are more than this number of interrupted connections from a host, that host is blocked from further connections. You can unblock blocked hosts with the `FLUSH HOSTS` statement. If a connection is established successfully within fewer than `max_connect_errors` [419] attempts after a previous connection was interrupted, the error count for the host is cleared to zero. However, once a host is blocked, the `FLUSH HOSTS` statement is the only way to unblock it.

- `max_connections` [419]

  The maximum permitted number of simultaneous client connections. By default, this is 100. See Section B.5.2.7, “`Too many connections`”, for more information.

  Increasing this value increases the number of file descriptors that `mysqld` requires. See Section 7.7.2, “How MySQL Opens and Closes Tables”, for comments on file descriptor limits.

- `max_delayed_threads` [419]

  Do not start more than this number of threads to handle `INSERT DELAYED` statements. If you try to insert data into a new table after all `INSERT DELAYED` threads are in use, the row is inserted as if the `DELAYED` attribute was not specified. If you set this to 0, MySQL never creates a thread to handle `DELAYED` rows; in effect, doing so disables `DELAYED` entirely. This variable was added in MySQL 3.23.0.

  For the `SESSION` value of this variable, the only valid values are 0 or the `GLOBAL` value.

- `max_error_count` [419]

  The maximum number of error, warning, and note messages to be stored for display by the `SHOW ERRORS` or `SHOW WARNINGS` statements. This variable was added in MySQL 4.1.0.

- `max_heap_table_size` [419]

  This variable sets the maximum size to which user-created `MEMORY` (`HEAP`) tables are permitted to grow. The value of the variable is used to calculate `MEMORY` table `MAX_ROWS` values. Setting this variable has no effect on any existing `MEMORY` table, unless the table is re-created with a statement such as `CREATE TABLE`, or altered with `ALTER TABLE` or `TRUNCATE TABLE`. A server restart also sets the maximum size of existing `MEMORY` tables to the global `max_heap_table_size` [419] value. This variable was added in MySQL 3.23.0.

  This variable is also used in conjunction with `tmp_table_size` [432] to limit the size of internal in-memory tables. See Section 7.7.4, “How MySQL Uses Internal Temporary Tables”.

- `max_insert_delayed_threads` [419]

  This variable is a synonym for `max_delayed_threads` [419]. It was added in MySQL 4.0.19.

- `max_join_size` [419]

  Do not permit statements that probably need to examine more than `max_join_size` [419] rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than `max_join_size` [419] disk seeks. By setting this value, you can catch statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a `WHERE` clause, that take a long time, or that return millions of rows.

Setting this variable to a value other than `DEFAULT` resets the value of `sql_big_selects` [428] to `0`. If you set the `sql_big_selects` [428] value again, the `max_join_size` [419] variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

This variable previously was named `sql_max_join_size`.

- `max_length_for_sort_data` [420]

  The cutoff on the size of index values that determines which `filesort` algorithm to use. See Section 7.3.1.7, "`ORDER BY` Optimization". This variable was added in MySQL 4.1.1

- `max_prepared_stmt_count` [420]

  This variable limits the total number of prepared statements in the server. It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. The default value is 16,382. The permissible range of values is from 0 to 1 million. Setting the value to 0 disables prepared statements. This variable was added in MySQL 4.1.19.

- `max_relay_log_size` [420]

  If a write by a replication slave to its relay log causes the current log file size to exceed the value of this variable, the slave rotates the relay logs (closes the current file and opens the next one). If `max_relay_log_size` [420] is 0, the server uses `max_binlog_size` [1184] for both the binary log and the relay log. If `max_relay_log_size` [420] is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set `max_relay_log_size` [420] to between 4096 bytes and 1GB (inclusive), or to `0`. The default value is `0`. This variable was added in MySQL 4.0.14. See Section 14.3, "Replication Implementation Details".

- `max_seeks_for_key` [420]

  Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see Section 12.4.5.13, "`SHOW INDEX` Syntax"). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

  This variable was added in MySQL 4.0.14.

- `max_sort_length` [420]

  The number of bytes to use when sorting `BLOB` or `TEXT` values. Only the first `max_sort_length` [420] bytes of each value are used; the rest are ignored.

- `max_tmp_tables` [420]

  The maximum number of temporary tables a client can keep open at the same time. (This variable does not yet do anything.)

- `max_user_connections` [420]

  The maximum number of simultaneous connections permitted to any given MySQL user account. A value of `0` means "no limit." This variable was added in MySQL 3.23.34.

This variable has only a global form.

- `max_write_lock_count` [421]

  After this many write locks, permit some pending read lock requests to be processed in between. This variable was added in MySQL 3.23.7.

- `myisam_data_pointer_size` [421]

  The default pointer size in bytes, to be used by `CREATE TABLE` for `MyISAM` tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 7. The default value is `4`. This variable was added in MySQL 4.1.2. See Section B.5.2.12, “`The table is full`”.

- `myisam_max_extra_sort_file_size` [421]

  If the temporary file used for fast `MyISAM` index creation would be larger than using the key cache by the amount specified here, prefer the key cache method. This is mainly used to force long character keys in large tables to use the slower key cache method to create the index. This variable was added in MySQL 3.23.37.

  Note

  The value is given in megabytes before 4.0.3 and in bytes thereafter.

- `myisam_max_sort_file_size` [421]

  The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. This variable was added in MySQL 3.23.37.

  Note

  The value is given in megabytes before 4.0.3 and in bytes thereafter.

  The default value is 2GB. If `MyISAM` index files exceed this size and disk space is available, increasing the value may help performance. The space must be available in the file system containing the directory where the original index file is located.

- `myisam_recover_options` [421]

  The value of the `--myisam-recover` [390] option. See Section 5.1.2, “Server Command Options”. This variable was added in MySQL 3.23.36.

- `myisam_repair_threads` [421]

  If this value is greater than 1, `MyISAM` table indexes are created in parallel (each index in its own thread) during the `Repair by sorting` process. The default value is 1.

  Note

  Multi-threaded repair is still *beta-quality* code. This variable was added in MySQL 4.0.13.

- `myisam_sort_buffer_size` [421]

The size of the buffer that is allocated when sorting MyISAM indexes during a REPAIR TABLE or when creating indexes with CREATE INDEX or ALTER TABLE. This variable was added in MySQL 3.23.16.

- myisam_stats_method [422]

  How the server treats NULL values when collecting statistics about the distribution of index values for MyISAM tables. This variable has three possible values, nulls_equal, nulls_unequal, and nulls_ignored. For nulls_equal, all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values. For nulls_unequal, NULL values are considered unequal, and each NULL forms a distinct value group of size 1. For nulls_ignored, NULL values are ignored.

  The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in Section 7.4.4, “MyISAM Index Statistics Collection”.

  Any unique prefix of a valid value may be used to set the value of this variable.

  This variable was added in MySQL 4.1.15. For older versions, the statistics collection method is equivalent to nulls_equal.

- named_pipe [422]

  On Windows, indicates whether the server supports connections over named pipes. This variable was added in MySQL 3.23.50.

- net_buffer_length [422]

  Each client thread is associated with a connection buffer and result buffer. Both begin with a size given by net_buffer_length [422] but are dynamically enlarged up to max_allowed_packet [418] bytes as needed. The result buffer shrinks to net_buffer_length [422] after each SQL statement.

  This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value to which net_buffer_length [422] can be set is 1MB.

- net_read_timeout [422]

  The number of seconds to wait for more data from a connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made through Unix socket files, named pipes, or shared memory. When the server is reading from the client, net_read_timeout [422] is the timeout value controlling when to abort. When the server is writing to the client, net_write_timeout [422] is the timeout value controlling when to abort. See also slave_net_timeout [1181]. This variable was added in MySQL 3.23.20.

- net_retry_count [422]

  If a read on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads. This variable was added in MySQL 3.23.7.

- net_write_timeout [422]

  The number of seconds to wait for a block to be written to a connection before aborting the write. This timeout applies only to TCP/IP connections, not to connections made using Unix socket files, named pipes, or shared memory. See also net_read_timeout [422]. This variable was added in MySQL 3.23.20.

- `new` [390]

  This variable is used in MySQL 4.0 to turn on some 4.1 behaviors. This variable was added in MySQL 4.0.12.

- `old_passwords` [423]

  Whether the server should use pre-4.1-style passwords for MySQL user accounts. This variable was added in MySQL 4.1.1.

- `one_shot` [423]

  This is not a variable, but it can be used when setting some variables. It is described in Section 12.4.4, "`SET` Syntax".

- `open_files_limit` [423]

  The number of files that the operating system permits `mysqld` to open. This is the real value permitted by the system and might be different from the value you gave using the `--open-files-limit` [391] option to `mysqld` or `mysqld_safe`. The value is 0 on systems where MySQL cannot change the number of open files. This variable was added in MySQL 3.23.20.

- `pid_file` [423]

  The path name of the process ID (PID) file. This variable can be set with the `--pid-file` [391] option. This variable was added in MySQL 3.23.23.

- `plugin_dir` [423]

  The path name of the plugin directory. This variable was added in MySQL 4.1.25. If the value is nonempty, user-defined function object files must be located in this directory. If the value is empty, the behavior that is used before 4.1.25 applies: The UDF object files must be located in a directory that is searched by your system's dynamic linker.

  If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` [423] read only to the server.

- `port` [423]

  The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` [391] option.

- `preload_buffer_size` [423]

  The size of the buffer that is allocated when preloading indexes. This variable was added in MySQL 4.1.1.

- `prepared_stmt_count` [423]

  The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` [420] system variable.) This variable was added in MySQL 4.1.19. In MySQL 4.1.23, it was converted to the global `Prepared_stmt_count` [453] status variable.

- `protocol_version` [423]

  The version of the client/server protocol used by the MySQL server. This variable was added in MySQL 3.23.18.

- `pseudo_thread_id` [424]

System Variable Name	`pseudo_thread_id` [424]	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`

This variable is for internal server use.

- `query_alloc_block_size` [424]

The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this parameter. This variable was added in MySQL 4.0.16.

- `query_cache_limit` [424]

Do not cache results that are larger than this number of bytes. The default value is 1MB. This variable was added in MySQL 4.0.1.

- `query_cache_min_res_unit` [424]

The minimum size for blocks allocated by the query cache. The default value is 4KB. Tuning information for this variable is given in Section 7.5.3.3, “Query Cache Configuration”. This variable is present from MySQL 4.1.

- `query_cache_size` [424]

The amount of memory allocated for caching query results. The default value is `0`, which disables the query cache. The permissible values are multiples of 1024; other values are rounded down to the nearest multiple. Note that `query_cache_size` [424] bytes of memory are allocated even if `query_cache_type` [424] is set to `0`. This variable was added in MySQL 4.0.1.

The query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value of `query_cache_size` [424] too small, a warning will occur, as described in Section 7.5.3.3, “Query Cache Configuration”.

- `query_cache_type` [424]

Set the query cache type. Setting the `GLOBAL` value sets the type for all clients that connect thereafter. Individual clients can set the `SESSION` value to affect their own use of the query cache.

Option	Description
`0` or `OFF`	Do not cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set `query_cache_size` [424] to 0.
`1` or `ON`	Cache all cacheable query results except for those that begin with `SELECT SQL_NO_CACHE`.
`2` or `DEMAND`	Cache results only for cacheable queries that begin with `SELECT SQL_CACHE`.

This variable defaults to `ON`.

Any unique prefix of a valid value may be used to set the value of this variable.

This variable was added in MySQL 4.0.3.

- `query_cache_wlock_invalidate` [425]

  Normally, when one client acquires a `WRITE` lock on a `MyISAM` table, other clients are not blocked from issuing statements that read from the table if the query results are present in the query cache. Setting this variable to 1 causes acquisition of a `WRITE` lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect. This variable was added in MySQL 4.0.19.

- `query_prealloc_size` [425]

  The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger `query_prealloc_size` [425] value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

  This variable was added in MySQL 4.0.16.

- `rand_seed1` [425]

  The `rand_seed1` [425] and `rand_seed2` [425] variables exist as session variables only, and can be set but not read. They are not shown in the output of `SHOW VARIABLES`. These two variables were added in MySQL 4.0.5.

  The purpose of these variables is to support replication of the `RAND()` [822] function. For statements that invoke `RAND()` [822], the master passes two values to the slave, where they are used to seed the random number generator. The slave uses these values to set the session variables `rand_seed1` [425] and `rand_seed2` [425] so that `RAND()` [822] on the slave generates the same value as on the master.

- `rand_seed2` [425]

  See the description for `rand_seed1` [425].

- `range_alloc_block_size` [425]

  The size of blocks that are allocated when doing range optimization. This variable was added in MySQL 4.0.16.

- `read_buffer_size` [425]

  Each thread that does a sequential scan allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value.

  `read_buffer_size` [425] and `read_rnd_buffer_size` [426] are not specific to any storage engine and apply in a general manner for optimization. See Section 7.8.4, “How MySQL Uses Memory”, for example.

  Before MySQL 4.0.3, this variable was named `record_buffer`.

- `read_only` [425]

  This variable is off by default. When it is enabled, the server permits no updates except from users that have the `SUPER` [493] privilege or (on a slave server) from updates performed by slave threads. On a slave server, this can be useful to ensure that the slave accepts updates only from its master server and not from clients.

`read_only` [425] exists only as a `GLOBAL` variable, so changes to its value require the `SUPER` [493] privilege. Changes to `read_only` [425] on a master server are not replicated to slave servers. The value can be set on a slave server independent of the setting on the master.

This variable was added in MySQL 4.0.14.

- `read_rnd_buffer_size` [426]

  When reading rows in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. See Section 7.3.1.7, "`ORDER BY` Optimization". Setting the variable to a large value can improve `ORDER BY` performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

  `read_buffer_size` [425] and `read_rnd_buffer_size` [426] are not specific to any storage engine and apply in a general manner for optimization. See Section 7.8.4, "How MySQL Uses Memory", for example.

  Before MySQL 4.0.3, this variable was named `record_rnd_buffer`.

- `relay_log_purge` [426]

  Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (`ON`).

- `relay_log_space_limit` [426]

  The maximum amount of space to use for all relay logs.

- `safe_show_database` [426]

  Do not show databases for which the user has no database or table privileges. This can improve security if you are concerned about people being able to see what databases other users have. See also `skip_show_database` [427].

  This variable was removed in MySQL 4.0.5. Beginning with this version, you should instead use the `SHOW DATABASES` [492] privilege to control access by MySQL accounts to databases.

- `secure_auth` [426]

  If the MySQL server has been started with the `--secure-auth` [392] option, it blocks connections from all accounts that have passwords stored in the old (pre-4.1) format. In that case, the value of this variable is `ON`, otherwise it is `OFF`.

  You should enable this option if you want to prevent all use of passwords in the old format (and hence insecure communication over the network). This variable was added in MySQL 4.1.1.

  Server startup fails with an error if this option is enabled and the privilege tables are in pre-4.1 format.

- `server_id` [426]

  The server ID, used in replication to give each master and slave a unique identity. This variable is set by the `--server-id` [1167] option. For each server participating in replication, you should pick a positive integer in the range from 1 to 2^{32} – 1 to act as that server's ID.

- `shared_memory` [426]

(Windows only.) Whether the server permits shared-memory connections. This variable was added in MySQL 4.1.1.

- `shared_memory_base_name` [427]

  (Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. This variable was added in MySQL 4.1.0.

- `skip_external_locking` [427]

  This is `OFF` if `mysqld` uses external locking, `ON` if external locking is disabled. Before MySQL 4.0.3, this variable was named `skip_locking`.

- `skip_networking` [427]

  This is `ON` if the server permits only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. On NetWare, only TCP/IP connections are supported, so do not set this variable to `ON`. This variable can be set to `ON` with the `--skip-networking` [393] option. This variable was added in MySQL 3.22.23.

- `skip_show_database` [427]

  This prevents people from using the `SHOW DATABASES` statement if they do not have the `SHOW DATABASES` [492] privilege. This can improve security if you are concerned about people being able to see what databases other users have. See also `safe_show_database` [426]. This variable was added in MySQL 3.23.4. As of MySQL 4.0.2, its effect also depends on the `SHOW DATABASES` [492] privilege: If the variable value is `ON`, the `SHOW DATABASES` statement is permitted only to users who have the `SHOW DATABASES` [492] privilege, and the statement displays all database names. If the value is `OFF`, `SHOW DATABASES` is permitted to all users, but displays each database name only if the user has the `SHOW DATABASES` [492] privilege or some privilege for the database. (Note that *any* global privilege is considered a privilege for the database.)

- `slow_launch_time` [427]

  If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` [455] status variable. This variable was added in MySQL 3.23.15.

- `socket` [427]

  On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

  On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is `MySQL` (not case sensitive).

- `sort_buffer_size` [427]

  Each session that needs to do a sort allocates a buffer of this size. `sort_buffer_size` [427] is not specific to any storage engine and applies in a general manner for optimization. See Section 7.3.1.7, “`ORDER BY` Optimization”, for example.

  If you see many `Sort_merge_passes` [455] per second in `SHOW GLOBAL STATUS` output, you can consider increasing the `sort_buffer_size` [427] value to speed up `ORDER BY` or `GROUP BY` operations that cannot be improved with query optimization or improved indexing. The entire buffer is allocated even if it is not all needed, so setting it larger than required globally will slow down most

queries that sort. It is best to increase it as a session setting, and only for the sessions that need a larger size. On Linux, there are thresholds of 256KB and 2MB where larger values may significantly slow down memory allocation, so you should consider staying below one of those values. Experiment to find the best value for your workload. See Section B.5.4.4, “Where MySQL Stores Temporary Files”.

- `sql_auto_is_null` [428]

System Variable Name	`sql_auto_is_null` [428]	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`1`

If this variable is set to 1 (the default), then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` [874] function. For details, including the return value after a multiple-row insert, see Section 11.13, “Information Functions”. If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` [783] comparison is used by some ODBC programs, such as Access. See Obtaining Auto-Increment Values. This behavior can be disabled by setting `sql_auto_is_null` [428] to 0.

- `sql_big_selects` [428]

System Variable Name	`sql_big_selects` [428]	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`1`

If set to 0, MySQL aborts `SELECT` statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of `max_join_size` [419]). This is useful when an inadvisable `WHERE` statement has been issued. The default value for a new connection is 1, which permits all `SELECT` statements.

If you set the `max_join_size` [419] system variable to a value other than `DEFAULT`, `sql_big_selects` [428] is set to 0.

- `sql_buffer_result` [428]

System Variable Name	`sql_buffer_result` [428]
Variable Scope	Session
Dynamic Variable	Yes

	Permitted Values	
	Type	`boolean`
	Default	`0`

If set to 1, `sql_buffer_result` [428] forces results from `SELECT` statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is 0. This variable was added in MySQL 3.23.13.

- `sql_log_bin` [429]

System Variable Name	`sql_log_bin` [429]	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`

If set to 0, no logging is done to the binary log for the client. The client must have the `SUPER` [493] privilege to set this option. The default value is 1. This variable was added in MySQL 3.23.16.

- `sql_log_off` [429]

System Variable Name	`sql_log_off` [429]	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`0`

If set to 1, no logging is done to the general query log for this client. The client must have the `SUPER` [493] privilege to set this option. The default value is 0.

- `sql_log_update` [429]

System Variable Name	`sql_log_update` [429]	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`

If set to `0`, no logging is done to the update log for the client. The client must have the `SUPER` [493] privilege to set this option. The default value is 1. This variable was added in MySQL 3.22.5.

- `sql_mode` [429]

The current server SQL mode. This variable was added in MySQL 3.23.41. It can be set dynamically as of MySQL 4.1.1. See Section 5.1.6, “Server SQL Modes”.

- `sql_notes` [429]

If set to 1 (the default), warnings of `Note` level are recorded. If set to 0, `Note` warnings are suppressed. `mysqldump` includes output to set this variable to 0 so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation. `sql_notes` [429] was added in MySQL 4.1.11.

- `sql_quote_show_create` [430]

  If set to 1 (the default), the server quotes identifiers for `SHOW CREATE TABLE` and `SHOW CREATE DATABASE` statements. If set to 0, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See Section 12.4.5.7, "`SHOW CREATE TABLE` Syntax", and Section 12.4.5.6, "`SHOW CREATE DATABASE` Syntax". This variable was added in MySQL 3.23.26.

- `sql_safe_updates` [430]

  If set to 1, MySQL aborts `UPDATE` or `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause. This makes it possible to catch `UPDATE` or `DELETE` statements where keys are not used properly and that would probably change or delete a large number of rows. The default value is 0. This variable was added in MySQL 3.22.32.

- `sql_select_limit` [430]

  The maximum number of rows to return from `SELECT` statements. The default value for a new connection is the maximum number of rows that the server permits per table, which depends on the server configuration and may be affected if the server build was configured with `--with-big-tables` [98]. Typical default values are (2^{32})–1 or (2^{64})–1. If you have changed the limit, the default value can be restored by assigning a value of `DEFAULT`.

  If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `sql_select_limit` [430].

  `sql_select_limit` [430] does not apply to `SELECT` statements executed within stored routines. It also does not apply to `SELECT` statements that do not produce a result set to be returned to the client. These include `SELECT` statements in subqueries, `CREATE TABLE ... SELECT`, and `INSERT INTO ... SELECT`.

- `sql_warnings` [430]

  This variable controls whether single-row `INSERT` statements produce an information string if warnings occur. The default is 0. Set the value to 1 to produce an information string. This variable was added in MySQL 3.22.11.

- `storage_engine` [430]

  This variable is a synonym for `table_type` [431]. It was added in MySQL 4.1.2.

- `sync_frm` [430]

  If this variable is set to 1, when any nontemporary table is created its `.frm` file is synchronized to disk (using `fdatasync()`). This is slower but safer in case of a crash. The default is 1. This was added as a command-line option in MySQL 4.0.18. It is also a settable global variable as of MySQL 4.1.3.

- `system_time_zone` [430]

  The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set `system_time_zone` [430]. Typically the time zone is

specified by the TZ environment variable. It also can be specified using the --timezone [245] option of the mysqld_safe script.

The system_time_zone [430] variable differs from time_zone [431]. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See Section 9.7, “MySQL Server Time Zone Support”.

system_time_zone [430] was added in MySQL 4.1.3.

- table_cache [431]

  The number of open tables for all threads. Increasing this value increases the number of file descriptors that mysqld requires. You can check whether you need to increase the table cache by checking the Opened_tables [453] status variable. See Section 5.1.5, “Server Status Variables”. If the value of Opened_tables [453] is large and you do not do FLUSH TABLES often (which just forces all tables to be closed and reopened), then you should increase the value of the table_cache [431] variable. For more information about the table cache, see Section 7.7.2, “How MySQL Opens and Closes Tables”.

- table_type [431]

  The default table type (storage engine). To set the table type at server startup, use the --default-table-type [386] option. This variable was added in MySQL 3.23.0. See Section 5.1.2, “Server Command Options”.

- thread_cache_size [431]

  How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than thread_cache_size [431] threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. Normally, this does not provide a notable performance improvement if you have a good thread implementation. However, if your server sees hundreds of connections per second you should normally set thread_cache_size [431] high enough so that most new connections use cached threads. By examining the difference between the Connections [450] and Threads_created [457] status variables, you can see how efficient the thread cache is. For details, see Section 5.1.5, “Server Status Variables”. This variable was added in MySQL 3.23.16.

- thread_concurrency [431]

  This variable is specific to Solaris systems, for which mysqld invokes the thr_setconcurrency() with the variable value. This function enables applications to give the threads system a hint about the desired number of threads that should be run at the same time. This variable was added in MySQL 3.23.7.

- thread_stack [431]

  The stack size for each thread. Many of the limits detected by the crash-me test are dependent on this value. The default is large enough for normal operation. See Section 7.1.3, “The MySQL Benchmark Suite”. The default is 64KB before MySQL 4.0.10 and 192KB thereafter. If the thread stack size is too small, it limits the complexity of the SQL statements that the server can handle, the recursion depth of stored procedures, and other memory-consuming actions.

- time_format [431]

  This variable is unused.

- time_zone [431]

The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is `'SYSTEM'` (which means, "use the value of `system_time_zone` [430]"). The value can be specified explicitly at server startup with the `--default-time-zone` [386] option. See Section 9.7, "MySQL Server Time Zone Support". This variable was added in MySQL 4.1.3.

- `timestamp = {timestamp_value | DEFAULT}` [432]

  Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. *`timestamp_value`* should be a Unix epoch timestamp, not a MySQL timestamp.

- `timezone` [432]

  The time zone for the server. This is set from the `TZ` environment variable when `mysqld` is started. The time zone also can be set by giving a `--timezone` [245] argument to `mysqld_safe`. This variable was added in MySQL 3.23.15. As of MySQL 4.1.3, it is obsolete and has been replaced by the `system_time_zone` [430] variable. See Section B.5.4.6, "Time Zone Problems".

- `tmp_table_size` [432]

  The maximum size of internal in-memory temporary tables. (The actual limit is determined as the minimum of `tmp_table_size` [432] and `max_heap_table_size` [419].) If an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk `MyISAM` table. Increase the value of `tmp_table_size` [432] (and `max_heap_table_size` [419] if necessary) if you do many advanced `GROUP BY` queries and you have lots of memory. This variable does not apply to user-created `MEMORY` tables.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` [450] and `Created_tmp_tables` [451] variables.

  See also Section 7.7.4, "How MySQL Uses Internal Temporary Tables".

- `tmpdir` [432]

  The directory used for temporary files and temporary tables. Starting from MySQL 4.1, this variable can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters ("`:`") on Unix and semicolon characters ("`;`") on Windows, NetWare, and OS/2.

  The multiple-directory feature can be used to spread the load between several physical disks. If the MySQL server is acting as a replication slave, you should not set `tmpdir` [432] to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails. However, if you are using MySQL 4.0.0 or later, you can set the slave's temporary directory using the `slave_load_tmpdir` [1180] variable. In that case, the slave will not use the general `tmpdir` [432] value and you can set `tmpdir` [432] to a nonpermanent location.

  This variable was added in MySQL 3.22.4.

- `transaction_alloc_block_size` [432]

  The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of `transaction_prealloc_size` [432]. This variable was added in MySQL 4.0.16.

- `transaction_prealloc_size` [432]

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size` [432]. For every allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` [432] bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` [432] bytes.

By making `transaction_prealloc_size` [432] sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls. This variable was added in MySQL 4.0.16.

- `tx_isolation` [433]

  The default transaction isolation level. This variable was added in MySQL 4.0.3.

  This variable is set by the `SET TRANSACTION ISOLATION LEVEL` statement. See Section 12.3.6, "`SET TRANSACTION` Syntax". If you set `tx_isolation` [433] directly to an isolation level name that contains a space, the name should be enclosed within quotation marks, with the space replaced by a dash. For example:

```
SET tx_isolation = 'READ-COMMITTED';
```

  Any unique prefix of a valid value may be used to set the value of this variable.

- `unique_checks` [433]

System Variable Name	`unique_checks` [433]	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`1`

  If set to 1 (the default), uniqueness checks for secondary indexes in `InnoDB` tables are performed. If set to 0, storage engines are permitted to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to `InnoDB`. This variable was added in MySQL 3.23.52.

  Note that setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still permitted to check for them and issue duplicate-key errors if it detects them.

- `version` [433]

  The version number for the server.

- `version_bdb` [433]

  The `BDB` storage engine version. This variable was added in MySQL 3.23.31 with the name `bdb_version` [407] and renamed to `version_bdb` [433] in MySQL 4.1.1.

- `version_comment` [433]

System Variable Name	`version_comment` [433]
Variable Scope	Global

Dynamic Variable	No	
	Permitted Values	
	Type	`string`

The `configure` script has a `--with-comment` option that permits a comment to be specified when building MySQL. This variable contains the value of that comment. This variable was added in MySQL 4.0.17.

- `version_compile_machine` [434]

System Variable Name	`version_compile_machine` [434]	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`

The type of machine or architecture on which MySQL was built. This variable was added in MySQL 4.1.1.

- `version_compile_os` [434]

  The type of operating system on which MySQL was built. This variable was added in MySQL 4.0.19.

- `wait_timeout` [434]

  The number of seconds the server waits for activity on a noninteractive connection before closing it. This timeout applies only to TCP/IP and Unix socket file connections, not to connections made using named pipes, or shared memory.

  On thread startup, the session `wait_timeout` [434] value is initialized from the global `wait_timeout` [434] value or from the global `interactive_timeout` [414] value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE` connect option to `mysql_real_connect()`). See also `interactive_timeout` [414].

- `warning_count` [434]

  The number of errors, warnings, and notes that resulted from the last statement that generated messages. This variable is read only. See Section 12.4.5.26, “`SHOW WARNINGS` Syntax”.

  This variable was added in MySQL 4.1.0.

5.1.4 Using System Variables

The MySQL server maintains many system variables that indicate how it is configured. Section 5.1.3, “Server System Variables”, describes the meaning of these variables. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. As of MySQL 4.0.3, most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

Beginning with MySQL 4.0.3, the server maintains two kinds of system variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes all global variables to their default values. These defaults can be changed by options specified on the command line or in an option file. (See Section 4.2.3, “Specifying Program Options”.)
- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, the client's SQL mode is controlled by the session `sql_mode` [429] value, which is initialized when the client connects to the value of the global `sql_mode` [429] value.

System variable values can be set globally at server startup by using options on the command line or in an option file. When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3; that is, units of kilobytes, megabytes, or gigabytes, respectively. Thus, the following command starts the server with a query cache size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --query_cache_size=16M --max_allowed_packet=1G
```

Before MySQL 4.0.3, use this syntax instead:

```
mysqld --set-variable=query_cache_size=16M \
       --set-variable=max_allowed_packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
query_cache_size=16M
max_allowed_packet=1G
```

Or like this before MySQL 4.0.2:

```
[mysqld]
set-variable=query_cache_size=16M
set-variable=max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; `16M` and `16m` are equivalent, as are `1G` and `1g`.

If you want to restrict the maximum value to which a system variable can be set at runtime with the `SET` statement, you can specify this maximum by using an option of the form `--maximum-`*`var_name`*`=`*`value`* at server startup. For example, to prevent the value of `query_cache_size` [424] from being increased to more than 32MB at runtime, use the option `--maximum-query_cache_size=32M`. This feature is available as of MySQL 4.0.2.

Many system variables are dynamic and can be changed while the server runs by using the `SET` statement. For a list, see Section 5.1.4.2, “Dynamic System Variables”. To change a system variable with `SET`, refer to it as *`var_name`*, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by `GLOBAL` or `@@global.`. The `SUPER` [493] privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by `SESSION`, `@@session.`, or `@@`. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- `LOCAL` and `@@local.` are synonyms for `SESSION` and `@@session.`.
- If no modifier is present, `SET` changes the session variable.

A SET statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent GLOBAL or SESSION modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The *@@var_name* syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the SET GLOBAL statement).

To prevent incorrect usage, MySQL produces an error if you use SET GLOBAL with a variable that can only be used with SET SESSION or if you do not specify GLOBAL (or @@global.) when setting a global variable.

To set a SESSION variable to the GLOBAL value or a GLOBAL value to the compiled-in MySQL default value, use the DEFAULT keyword. For example, the following two statements are identical in setting the session value of max_join_size [419] to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to DEFAULT. In such cases, use of DEFAULT results in an error.

You can refer to the values of specific global or sesson system variables in expressions by using one of the @@-modifiers. For example, you can retrieve values in a SELECT statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as *@@var_name* (that is, when you do not specify @@global. or @@session.), MySQL returns the session value if it exists and the global value otherwise. (This differs from SET *@@var_name = value*, which always refers to the session value.)

Note

Some variables displayed by SHOW VARIABLES may not be available using SELECT *@@var_name* syntax; an Unknown system variable occurs. As a workaround in such cases, you can use SHOW VARIABLES LIKE '*var_name*'.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with SET at runtime. On the other hand, with SET you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

Note

Some system variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` [410] works but `--delay_key_write=ON` [410] does not.

To display system variable names and values, use the `SHOW VARIABLES` statement:

```
mysql> SHOW VARIABLES;
+---------------------------------+-------------------------------------+
| Variable_name                   | Value                               |
+---------------------------------+-------------------------------------+
| back_log                        | 50                                  |
| basedir                         | /usr/local/mysql                    |
| bdb_cache_size                  | 8388600                             |
| bdb_home                        | /usr/local/mysql                    |
| bdb_log_buffer_size             | 32768                               |
| bdb_logdir                      |                                     |
| bdb_max_lock                    | 10000                               |
| bdb_shared_data                 | OFF                                 |
| bdb_tmpdir                      | /tmp/                               |
| binlog_cache_size               | 32768                               |
| bulk_insert_buffer_size         | 8388608                             |
| character_set_client            | latin1                              |
| character_set_connection        | latin1                              |
| character_set_database          | latin1                              |
| character_set_results           | latin1                              |
| character_set_server            | latin1                              |
| character_set_system            | utf8                                |
| character_sets_dir              | /usr/local/mysql/share/charsets/    |
| collation_connection            | latin1_swedish_ci                   |
| collation_database              | latin1_swedish_ci                   |
| collation_server                | latin1_swedish_ci                   |
...
| innodb_additional_mem_pool_size | 1048576                             |
| innodb_autoextend_increment     | 8                                   |
| innodb_buffer_pool_awe_mem_mb   | 0                                   |
| innodb_buffer_pool_size         | 8388608                             |
| innodb_data_file_path           | ibdata1:10M:autoextend              |
| innodb_data_home_dir            |                                     |
...
| version                         | 4.1.18-max-log                      |
| version_comment                 | MySQL Community Edition - Max (GPL) |
| version_compile_machine         | i686                                |
| version_compile_os              | pc-linux-gnu                        |
| wait_timeout                    | 28800                               |
+---------------------------------+-------------------------------------+
```

With a `LIKE` [804] clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a `LIKE` [804] clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the "%" wildcard character in a LIKE [804] clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because "_" is a wildcard that matches any single character, you should escape it as "_" to match it literally. In practice, this is rarely necessary.

For SHOW VARIABLES, if you specify neither GLOBAL nor SESSION, MySQL returns SESSION values.

The reason for requiring the GLOBAL keyword when setting GLOBAL-only variables but not when retrieving them is to prevent problems in the future. If we were to remove a SESSION variable that has the same name as a GLOBAL variable, a client with the SUPER [493] privilege might accidentally change the GLOBAL variable rather than just the SESSION variable for its own connection. If we add a SESSION variable with the same name as a GLOBAL variable, a client that intends to change the GLOBAL variable might find only its own SESSION variable changed.

5.1.4.1 Structured System Variables

Structured system variables are supported beginning with MySQL 4.1.1. A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

In MySQL 4.1 (4.1.1 and above), MySQL supports one structured variable type. It specifies parameters that govern the operation of key caches. A key cache structured variable has these components:

- key_buffer_size [415]
- key_cache_block_size [416]
- key_cache_division_limit [416]
- key_cache_age_threshold [416]

The purpose of this section is to describe the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in Section 7.5.1, "The MyISAM Key Cache".

To refer to a component of a structured variable instance, you can use a compound name in *instance_name.component_name* format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of default is always predefined. If you refer to a component of a structured variable without any instance name, the default instance is used. Thus, default.key_buffer_size and key_buffer_size [415] both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but `` `hot-cache` `` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as `@@global.`*`var_name`* for referring to nonstructured system variables.

At the moment, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules will assume greater significance if some other type of structured variable is created in the future.

With one exception, it is permissible to refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
[mysqld]
hot_cache.key_buffer_size=64K
```

If you start the server with such an option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `--default.key_buffer_size=256K`.) In addition, the server creates a second key cache named `extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
```

```
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE` [804] pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

5.1.4.2 Dynamic System Variables

Beginning with MySQL 4.0.3, many server system variables are dynamic and can be set at runtime using `SET GLOBAL` or `SET SESSION`. You can also select their values using `SELECT`. See Section 5.1.4, “Using System Variables”.

The following table shows the full list of all dynamic system variables. The last column indicates for each variable whether `GLOBAL` or `SESSION` (or both) apply. The table also lists session options that can be set with the `SET` statement. Section 5.1.3, “Server System Variables”, discusses these options.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, `ON` or `OFF`. (If you set them on the command line or in an option file, use the numeric values.) Variables that are marked as “enumeration” normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration value corresponds to 0. This differs from `ENUM` columns, for which the first enumeration value corresponds to 1.

Table 5.3 Dynamic Variable Summary

Variable Name	Variable Type	Variable Scope
`autocommit` [406]	boolean	`SESSION`
`big_tables` [384]	boolean	`SESSION`
`binlog_cache_size` [407]	numeric	`GLOBAL`
`bulk_insert_buffer_size` [408]	numeric	`GLOBAL` | `SESSION`
`character_set_client` [408]	string	`GLOBAL` | `SESSION`
`character_set_connection` [408]	string	`GLOBAL` | `SESSION`
`character_set_database` [408]	string	`GLOBAL` | `SESSION`
`character_set_results` [408]	string	`GLOBAL` | `SESSION`
`character_set_server` [385]	string	`GLOBAL` | `SESSION`
`collation_connection` [409]	string	`GLOBAL` | `SESSION`
`collation_database` [409]	string	`GLOBAL` | `SESSION`
`collation_server` [385]	string	`GLOBAL` | `SESSION`
`concurrent_insert` [409]	boolean	`GLOBAL`
`connect_timeout` [409]	numeric	`GLOBAL`
`debug` [385]	string	`GLOBAL` | `SESSION`

Variable Name	Variable Type	Variable Scope
storage_engine [386]	enumeration	GLOBAL | SESSION
default_week_format [410]	numeric	GLOBAL | SESSION
delay_key_write [386]	enumeration	GLOBAL
delayed_insert_limit [410]	numeric	GLOBAL
delayed_insert_timeout [410]	numeric	GLOBAL
delayed_queue_size [410]	numeric	GLOBAL
expire_logs_days [411]	numeric	GLOBAL
flush [411]	boolean	GLOBAL
flush_time [411]	numeric	GLOBAL
foreign_key_checks [411]	boolean	SESSION
ft_boolean_syntax [411]	string	GLOBAL
group_concat_max_len [412]	numeric	GLOBAL | SESSION
identity [414]	numeric	SESSION
init_connect [414]	string	GLOBAL
init_slave [1180]	string	GLOBAL
innodb_autoextend_increment [1067]	numeric	GLOBAL
innodb_fast_shutdown [1068]	numeric	GLOBAL
innodb_flush_log_at_trx_commit [1068]	enumeration	GLOBAL
innodb_max_dirty_pages_pct [1071]	numeric	GLOBAL
innodb_max_purge_lag [1071]	numeric	GLOBAL
innodb_table_locks [1072]	boolean	GLOBAL | SESSION
innodb_thread_concurrency [1072]	numeric	GLOBAL
insert_id [414]	numeric	SESSION
interactive_timeout [414]	numeric	GLOBAL | SESSION
join_buffer_size [415]	numeric	GLOBAL | SESSION
key_buffer_size [415]	numeric	GLOBAL
key_cache_age_threshold [416]	numeric	GLOBAL
key_cache_block_size [416]	numeric	GLOBAL
key_cache_division_limit [416]	numeric	GLOBAL
last_insert_id [416]	numeric	SESSION
lc_time_names [417]	string	GLOBAL | SESSION
local_infile [417]	boolean	GLOBAL
log_queries_not_using_indexes [388]	boolean	GLOBAL
log_warnings [389]	numeric	GLOBAL | SESSION
long_query_time [417]	numeric	GLOBAL | SESSION
low_priority_updates [389]	boolean	GLOBAL | SESSION
max_allowed_packet [418]	numeric	GLOBAL
max_binlog_cache_size [1183]	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
max_binlog_size [1184]	numeric	GLOBAL
max_connect_errors [419]	numeric	GLOBAL
max_connections [419]	numeric	GLOBAL
max_delayed_threads [419]	numeric	GLOBAL | SESSION
max_error_count [419]	numeric	GLOBAL | SESSION
max_heap_table_size [419]	numeric	GLOBAL | SESSION
max_insert_delayed_threads [419]	numeric	GLOBAL | SESSION
max_join_size [419]	numeric	GLOBAL | SESSION
max_length_for_sort_data [420]	numeric	GLOBAL | SESSION
max_prepared_stmt_count [420]	numeric	GLOBAL
max_relay_log_size [420]	numeric	GLOBAL
max_seeks_for_key [420]	numeric	GLOBAL | SESSION
max_sort_length [420]	numeric	GLOBAL | SESSION
max_user_connections [420]	numeric	GLOBAL
max_write_lock_count [421]	numeric	GLOBAL
myisam_data_pointer_size [421]	numeric	GLOBAL
myisam_max_sort_file_size [421]	numeric	GLOBAL
myisam_repair_threads [421]	numeric	GLOBAL | SESSION
myisam_sort_buffer_size [421]	numeric	GLOBAL | SESSION
myisam_stats_method [422]	enumeration	GLOBAL | SESSION
ndb_autoincrement_prefetch_sz [1302]	numeric	GLOBAL | SESSION
ndb_cache_check_time [1302]	numeric	GLOBAL
ndb_force_send [1303]	boolean	GLOBAL | SESSION
ndb_use_exact_count [1305]	boolean	GLOBAL | SESSION
ndb_use_transactions [1305]	boolean	GLOBAL | SESSION
net_buffer_length [422]	numeric	GLOBAL | SESSION
net_read_timeout [422]	numeric	GLOBAL | SESSION
net_retry_count [422]	numeric	GLOBAL | SESSION
net_write_timeout [422]	numeric	GLOBAL | SESSION
new [390]	boolean	GLOBAL | SESSION
old_passwords [423]	boolean	GLOBAL | SESSION
preload_buffer_size [423]	numeric	GLOBAL | SESSION
pseudo_thread_id [424]	numeric	SESSION
query_alloc_block_size [424]	numeric	GLOBAL | SESSION
query_cache_limit [424]	numeric	GLOBAL
query_cache_min_res_unit [424]	numeric	GLOBAL
query_cache_size [424]	numeric	GLOBAL
query_cache_type [424]	enumeration	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
query_cache_wlock_invalidate [425]	boolean	GLOBAL | SESSION
query_prealloc_size [425]	numeric	GLOBAL | SESSION
rand_seed1 [425]	numeric	SESSION
rand_seed2 [425]	numeric	SESSION
range_alloc_block_size [425]	numeric	GLOBAL | SESSION
read_buffer_size [425]	numeric	GLOBAL | SESSION
read_only [425]	boolean	GLOBAL
read_rnd_buffer_size [426]	numeric	GLOBAL | SESSION
relay_log_purge [426]	boolean	GLOBAL
rpl_recovery_rank [1180]	numeric	GLOBAL
safe_show_database [391]	boolean	GLOBAL
secure_auth [392]	boolean	GLOBAL
server_id [1167]	numeric	GLOBAL
slave_compressed_protocol [1180]	boolean	GLOBAL
slave_net_timeout [1179]	numeric	GLOBAL
slave_transaction_retries [1181]	numeric	GLOBAL
slow_launch_time [427]	numeric	GLOBAL
sort_buffer_size [427]	numeric	GLOBAL | SESSION
sql_auto_is_null [428]	boolean	SESSION
sql_big_selects [428]	boolean	SESSION
sql_big_tables [407]	boolean	SESSION
sql_buffer_result [428]	boolean	SESSION
sql_log_bin [429]	boolean	SESSION
sql_log_off [429]	boolean	SESSION
sql_log_update [429]	boolean	SESSION
sql_low_priority_updates [418]	boolean	GLOBAL | SESSION
sql_max_join_size [419]	numeric	GLOBAL | SESSION
sql_mode [394]	set	GLOBAL | SESSION
sql_notes [429]	boolean	SESSION
sql_quote_show_create [430]	boolean	SESSION
sql_safe_updates [430]	boolean	SESSION
sql_select_limit [430]	numeric	GLOBAL | SESSION
sql_slave_skip_counter [1181]	numeric	GLOBAL
sql_warnings [430]	boolean	SESSION
storage_engine [430]	enumeration	GLOBAL | SESSION
sync_binlog [1184]	numeric	GLOBAL
sync_frm [430]	boolean	GLOBAL
table_cache [431]	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
table_type [430]	enumeration	GLOBAL | SESSION
thread_cache_size [431]	numeric	GLOBAL
time_zone [431]	string	GLOBAL | SESSION
timestamp [432]	numeric	SESSION
tmp_table_size [432]	numeric	GLOBAL | SESSION
transaction_alloc_block_size [432]	numeric	GLOBAL | SESSION
transaction_prealloc_size [432]	numeric	GLOBAL | SESSION
tx_isolation [433]	enumeration	GLOBAL | SESSION
unique_checks [433]	boolean	SESSION
wait_timeout [434]	numeric	GLOBAL | SESSION

5.1.5 Server Status Variables

The server maintains many status variables that provide information about its operation. You can view these variables and their values by using the SHOW STATUS statement (see Section 12.4.5.22, "SHOW STATUS Syntax").

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
| Bytes_sent               | 1176560426 |
| Connections              | 30023      |
...
```

The following table lists all available server status variables:

Table 5.4 Status Variable Summary

Variable Name	Variable Type
Aborted_clients [449]	numeric
Aborted_connects [449]	numeric
Binlog_cache_disk_use [449]	numeric
Binlog_cache_use [449]	numeric
Bytes_received [449]	numeric
Bytes_sent [450]	numeric
Com_admin_commands [450]	numeric
Com_alter_db [450]	numeric
Com_alter_table [450]	numeric
Com_analyze [450]	numeric
Com_backup_table [450]	numeric
Com_begin [450]	numeric
Com_change_db [450]	numeric

Variable Name	Variable Type
Com_change_master [450]	numeric
Com_check [450]	numeric
Com_checksum [450]	numeric
Com_commit [450]	numeric
Com_create_db [450]	numeric
Com_create_index [450]	numeric
Com_create_table [450]	numeric
Com_dealloc_sql [450]	numeric
Com_delete [450]	numeric
Com_delete_multi [450]	numeric
Com_do [450]	numeric
Com_drop_db [450]	numeric
Com_drop_index [450]	numeric
Com_drop_table [450]	numeric
Com_drop_user [450]	numeric
Com_execute_sql [450]	numeric
Com_flush [450]	numeric
Com_grant [450]	numeric
Com_ha_close [450]	numeric
Com_ha_open [450]	numeric
Com_ha_read [450]	numeric
Com_help [450]	numeric
Com_insert [450]	numeric
Com_insert_select [450]	numeric
Com_kill [450]	numeric
Com_load [450]	numeric
Com_load_master_data [450]	numeric
Com_load_master_table [450]	numeric
Com_lock_tables [450]	numeric
Com_optimize [450]	numeric
Com_preload_keys [450]	numeric
Com_prepare_sql [450]	numeric
Com_rename_table [450]	numeric
Com_repair [450]	numeric
Com_replace [450]	numeric
Com_replace_select [450]	numeric
Com_reset [450]	numeric
Com_restore_table [450]	numeric

Variable Name	Variable Type
Com_revoke [450]	numeric
Com_revoke_all [450]	numeric
Com_rollback [450]	numeric
Com_savepoint [450]	numeric
Com_select [450]	numeric
Com_set_option [450]	numeric
Com_show_binlog_events [450]	numeric
Com_show_binlogs [450]	numeric
Com_show_charsets [450]	numeric
Com_show_collations [450]	numeric
Com_show_column_types [450]	numeric
Com_show_create_db [450]	numeric
Com_show_create_event [450]	numeric
Com_show_create_table [450]	numeric
Com_show_databases [450]	numeric
Com_show_engine_logs [450]	numeric
Com_show_engine_mutex [450]	numeric
Com_show_engine_status [450]	numeric
Com_show_errors [450]	numeric
Com_show_fields [450]	numeric
Com_show_grants [450]	numeric
Com_show_innodb_status [450]	numeric
Com_show_keys [450]	numeric
Com_show_logs [450]	numeric
Com_show_master_status [450]	numeric
Com_show_ndb_status [450]	numeric
Com_show_new_master [450]	numeric
Com_show_open_tables [450]	numeric
Com_show_privileges [450]	numeric
Com_show_processlist [450]	numeric
Com_show_slave_hosts [450]	numeric
Com_show_slave_status [450]	numeric
Com_show_status [450]	numeric
Com_show_storage_engines [450]	numeric
Com_show_tables [450]	numeric
Com_show_variables [450]	numeric
Com_show_warnings [450]	numeric
Com_slave_start [450]	numeric

Variable Name	Variable Type
Com_slave_stop [450]	numeric
Com_stmt_close [450]	numeric
Com_stmt_execute [450]	numeric
Com_stmt_fetch [450]	numeric
Com_stmt_prepare [450]	numeric
Com_stmt_reset [450]	numeric
Com_stmt_send_long_data [450]	numeric
Com_truncate [450]	numeric
Com_unlock_tables [450]	numeric
Com_update [450]	numeric
Com_update_multi [450]	numeric
Created_tmp_disk_tables [450]	numeric
Created_tmp_files [451]	numeric
Created_tmp_tables [451]	numeric
Handler_commit [451]	numeric
Handler_delete [451]	numeric
Handler_discover [1306]	numeric
Handler_read_first [451]	numeric
Handler_read_key [451]	numeric
Handler_read_next [451]	numeric
Handler_read_prev [451]	numeric
Handler_read_rnd [452]	numeric
Handler_read_rnd_next [452]	numeric
Handler_rollback [452]	numeric
Handler_update [452]	numeric
Handler_write [452]	numeric
Key_blocks_not_flushed [452]	numeric
Key_blocks_unused [452]	numeric
Key_blocks_used [452]	numeric
Key_read_requests [452]	numeric
Key_reads [452]	numeric
Key_write_requests [452]	numeric
Key_writes [452]	numeric
Max_used_connections [453]	numeric
Not_flushed_delayed_rows	numeric
Open_files	numeric
Open_streams [453]	numeric
Open_tables [453]	numeric

Variable Name	Variable Type
Opened_tables [453]	numeric
Prepared_stmt_count [453]	numeric
Qcache_free_blocks [453]	numeric
Qcache_free_memory [453]	numeric
Qcache_hits [453]	numeric
Qcache_inserts [453]	numeric
Qcache_lowmem_prunes [453]	numeric
Qcache_not_cached [453]	numeric
Qcache_queries_in_cache [454]	numeric
Qcache_total_blocks [454]	numeric
Questions [454]	numeric
Select_full_join [454]	numeric
Select_full_range_join [454]	numeric
Select_range [454]	numeric
Select_range_check [454]	numeric
Select_scan [454]	numeric
Slave_open_temp_tables [454]	numeric
Slow_launch_threads [455]	numeric
Slow_queries [455]	numeric
Sort_merge_passes [455]	numeric
Sort_range [455]	numeric
Sort_rows [455]	numeric
Sort_scan [455]	numeric
Ssl_accept_renegotiates [455]	numeric
Ssl_accepts [455]	numeric
Ssl_callback_cache_hits [455]	numeric
Ssl_cipher [455]	string
Ssl_cipher_list [455]	string
Ssl_client_connects [455]	numeric
Ssl_connect_renegotiates [455]	numeric
Ssl_ctx_verify_depth [456]	numeric
Ssl_ctx_verify_mode [456]	numeric
Ssl_default_timeout [456]	numeric
Ssl_finished_accepts [456]	numeric
Ssl_finished_connects [456]	numeric
Ssl_session_cache_hits [456]	numeric
Ssl_session_cache_misses [456]	numeric
Ssl_session_cache_mode [456]	string

Variable Name	Variable Type
Ssl_session_cache_overflows [456]	numeric
Ssl_session_cache_size [456]	numeric
Ssl_session_cache_timeouts [456]	numeric
Ssl_sessions_reused [456]	numeric
Ssl_used_session_cache_entries [456]	numeric
Ssl_verify_depth [456]	numeric
Ssl_verify_mode [456]	numeric
Ssl_version [457]	string
Table_locks_immediate [457]	numeric
Table_locks_waited [457]	numeric
Threads_cached [457]	numeric
Threads_connected [457]	numeric
Threads_created [457]	numeric
Threads_running [457]	numeric
Uptime [457]	numeric

Many status variables are reset to 0 by the FLUSH STATUS statement.

The status variables have the following meanings. The Com_*xxx* statement counter variables were added beginning with MySQL 3.23.47. The Qcache_*xxx* query cache variables were added beginning with MySQL 4.0.1. Otherwise, variables with no version indicated have been present since at least MySQL 3.22.

For meanings of status variables specific to MySQL Cluster, see Section 15.3.4.4, “MySQL Cluster Status Variables”.

- Aborted_clients [449]

  The number of connections that were aborted because the client died without closing the connection properly. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

- Aborted_connects [449]

  The number of failed attempts to connect to the MySQL server. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

- Binlog_cache_disk_use [449]

  The number of transactions that used the temporary binary log cache but that exceeded the value of binlog_cache_size [407] and used a temporary file to store statements from the transaction. This variable was added in MySQL 4.1.2.

- Binlog_cache_use [449]

  The number of transactions that used the temporary binary log cache. This variable was added in MySQL 4.1.2.

- Bytes_received [449]

  The number of bytes received from all clients. This variable was added in MySQL 3.23.7.

- `Bytes_sent` [450]

  The number of bytes sent to all clients. This variable was added in MySQL 3.23.7.

- `Com_xxx`

  The `Com_xxx` statement counter variables were added beginning with MySQL 3.23.47. They indicate the number of times each *`xxx`* statement has been executed. There is one status variable for each type of statement. For example, `Com_delete` and `Com_insert` count `DELETE` and `INSERT` statements, respectively. However, if a query result is returned from query cache, the server increments the `Qcache_hits` [453] status variable, not `Com_select`. See Section 7.5.3.4, "Query Cache Status and Maintenance".

  New `Com_stmt_xxx` status variables have been added in MySQL 4.1.13:

  - `Com_stmt_prepare`
  - `Com_stmt_execute`
  - `Com_stmt_send_long_data`
  - `Com_stmt_reset`
  - `Com_stmt_close`

  Those variables stand for prepared statement commands. Their names refer to the `COM_xxx` command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as `mysql_stmt_prepare()`, `mysql_stmt_execute()`, and so forth are executed. However, `Com_stmt_prepare`, `Com_stmt_execute` and `Com_stmt_close` also increase for `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE`, respectively. Additionally, the values of the older (available since MySQL 4.1.3) statement counter variables `Com_prepare_sql`, `Com_execute_sql`, and `Com_dealloc_sql` increase for the `PREPARE`, `EXECUTE`, and `DEALLOCATE PREPARE` statements.

  All of the `Com_stmt_xxx` variables are increased even if their argument (a prepared statement) is unknown or an error occurred during execution; in other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

- `Connections` [450]

  The number of connection attempts (successful or not) to the MySQL server.

- `Created_tmp_disk_tables` [450]

  The number of internal on-disk temporary tables created by the server while executing statements. This variable was added in MySQL 3.23.24.

  If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the `tmp_table_size` [432] and `max_heap_table_size` [419] values. If `Created_tmp_disk_tables` [450] is large, you may want to increase the `tmp_table_size` [432] or `max_heap_table_size` [419] value to lessen the likelihood that internal temporary tables in memory will be converted to on-disk tables.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` [450] and `Created_tmp_tables` [451] variables.

See also Section 7.7.4, “How MySQL Uses Internal Temporary Tables”.

- `Created_tmp_files` [451]

  How many temporary files `mysqld` has created. This variable was added in MySQL 3.23.28.

- `Created_tmp_tables` [451]

  The number of internal temporary tables created by the server while executing statements.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` [450] and `Created_tmp_tables` [451] variables.

  See also Section 7.7.4, “How MySQL Uses Internal Temporary Tables”.

- `Delayed_errors` [451]

  The number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).

- `Delayed_insert_threads` [451]

  The number of `INSERT DELAYED` handler threads in use.

- `Delayed_writes` [451]

  The number of `INSERT DELAYED` rows written.

- `Flush_commands` [451]

  The number of executed `FLUSH` statements.

- `Handler_commit` [451]

  The number of internal `COMMIT` statements. This variable was added in MySQL 4.0.2.

- `Handler_delete` [451]

  The number of times a row was deleted from a table.

- `Handler_read_first` [451]

  The number of times the first entry in an index was read. If this value is high, it suggests that the server is doing a lot of full index scans; for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed.

- `Handler_read_key` [451]

  The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- `Handler_read_next` [451]

  The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev` [451]

The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`. This variable was added in MySQL 3.23.6.

- `Handler_read_rnd` [452]

  The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.

- `Handler_read_rnd_next` [452]

  The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback` [452]

  The number of internal `ROLLBACK` statements. This variable was added in MySQL 4.0.2.

- `Handler_update` [452]

  The number of requests to update a row in a table.

- `Handler_write` [452]

  The number of requests to insert a row in a table.

- `Key_blocks_not_flushed` [452]

  The number of key blocks in the key cache that have changed but have not yet been flushed to disk. This variable was added in MySQL 4.1.1. It used to be known as `Not_flushed_key_blocks` [453].

- `Key_blocks_unused` [452]

  The number of unused blocks in the key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` [415] in Section 5.1.3, “Server System Variables”. This variable was added in MySQL 4.1.2.

- `Key_blocks_used` [452]

  The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- `Key_read_requests` [452]

  The number of requests to read a key block from the cache.

- `Key_reads` [452]

  The number of physical reads of a key block from disk. If `Key_reads` [452] is large, then your `key_buffer_size` [415] value is probably too small. The cache miss rate can be calculated as `Key_reads` [452]/`Key_read_requests` [452].

- `Key_write_requests` [452]

  The number of requests to write a key block to the cache.

- `Key_writes` [452]

The number of physical writes of a key block to disk.

- `Max_used_connections` [453]

  The maximum number of connections that have been in use simultaneously since the server started.

- `Not_flushed_delayed_rows`

  The number of rows waiting to be written in `INSERT DELAYED` queues.

- `Not_flushed_key_blocks` [453]

  The old name for `Key_blocks_not_flushed` [452] before MySQL 4.1.1.

- `Open_files`

  The number of files that are open. This count includes regular files opened by the server. It does not include other types of files such as sockets or pipes. Also, the count does not include files that storage engines open using their own internal functions rather than asking the server level to do so.

- `Open_streams` [453]

  The number of streams that are open (used mainly for logging).

- `Open_tables` [453]

  The number of tables that are open.

- `Opened_tables` [453]

  The number of tables that have been opened. If `Opened_tables` [453] is big, your `table_cache` [431] value is probably too small.

- `Prepared_stmt_count` [453]

  The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` [420] system variable.) This variable was added in MySQL 4.1.23.

- `Qcache_free_blocks` [453]

  The number of free memory blocks in the query cache.

- `Qcache_free_memory` [453]

  The amount of free memory for the query cache.

- `Qcache_hits` [453]

  The number of query cache hits.

- `Qcache_inserts` [453]

  The number of queries added to the query cache.

- `Qcache_lowmem_prunes` [453]

  The number of queries that were deleted from the query cache because of low memory.

- `Qcache_not_cached` [453]

The number of noncached queries (not cacheable, or not cached due to the `query_cache_type` [424] setting).

- `Qcache_queries_in_cache` [454]

  The number of queries registered in the query cache.

- `Qcache_total_blocks` [454]

  The total number of blocks in the query cache.

- `Questions` [454]

  The number of statements that clients have sent to the server.

- `Rpl_status` [454]

  The status of fail-safe replication (not implemented).

- `Select_full_join` [454]

  The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables. This variable was added in MySQL 3.23.25.

- `Select_full_range_join` [454]

  The number of joins that used a range search on a reference table. This variable was added in MySQL 3.23.25.

- `Select_range` [454]

  The number of joins that used ranges on the first table. This is normally not critical issue even if the value is quite large. This variable was added in MySQL 3.23.25.

- `Select_range_check` [454]

  The number of joins without keys that check for key usage after each row. (If this is not equal to `0`, you should very carefully check the indexes of your tables.) This variable was added in MySQL 3.23.25.

- `Select_scan` [454]

  The number of joins that did a full scan of the first table. This variable was added in MySQL 3.23.25.

- `Slave_open_temp_tables` [454]

  The number of temporary tables that the slave SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the slave; see Section 14.7.12, “Replication and Temporary Tables”. This variable was added in MySQL 3.23.29.

- `Slave_retried_transactions` [454]

  Total (since startup) number of times the replication slave SQL thread has retried transactions. This variable was added in MySQL 4.1.11.

- `Slave_running`

  This is `ON` if this server is a replication slave that is connected to a replication master, and both the I/O and SQL threads are running; otherwise, it is `OFF`.

This variable was added in MySQL 3.23.16.

- `Slow_launch_threads` [455]

  The number of threads that have taken more than `slow_launch_time` [427] seconds to create. This variable was added in MySQL 3.23.15.

- `Slow_queries` [455]

  The number of queries that have taken more than `long_query_time` [417] seconds. See Section 5.3.5, “The Slow Query Log”.

- `Sort_merge_passes` [455]

  The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the `sort_buffer_size` [427] system variable. This variable was added in MySQL 3.23.28.

- `Sort_range` [455]

  The number of sorts that were done with ranges. This variable was added in MySQL 3.23.25.

- `Sort_rows` [455]

  The number of sorted rows. This variable was added in MySQL 3.23.25.

- `Sort_scan` [455]

  The number of sorts that were done by scanning the table. This variable was added in MySQL 3.23.25.

- `Ssl_accept_renegotiates` [455]

  The number of negotiates needed to establish the connection. This variable was added in MySQL 4.0.0.

- `Ssl_accepts` [455]

  The number of accepted SSL connections. This variable was added in MySQL 4.0.0.

- `Ssl_callback_cache_hits` [455]

  The number of callback cache hits. This variable was added in MySQL 4.0.0.

- `Ssl_cipher` [455]

  The current SSL cipher (empty for non-SSL connections). This variable was added in MySQL 4.0.0.

- `Ssl_cipher_list` [455]

  The list of possible SSL ciphers. This variable was added in MySQL 4.0.0.

- `Ssl_client_connects` [455]

  The number of SSL connection attempts to an SSL-enabled master. This variable was added in MySQL 4.0.0.

- `Ssl_connect_renegotiates` [455]

  The number of negotiates needed to establish the connection to an SSL-enabled master. This variable was added in MySQL 4.0.0.

- `Ssl_ctx_verify_depth` [456]

  The SSL context verification depth (how many certificates in the chain are tested). This variable was added in MySQL 4.0.0.

- `Ssl_ctx_verify_mode` [456]

  The SSL context verification mode. This variable was added in MySQL 4.0.0.

- `Ssl_default_timeout` [456]

  The default SSL timeout. This variable was added in MySQL 4.0.0.

- `Ssl_finished_accepts` [456]

  The number of successful SSL connections to the server. This variable was added in MySQL 4.0.0.

- `Ssl_finished_connects` [456]

  The number of successful slave connections to an SSL-enabled master. This variable was added in MySQL 4.0.0.

- `Ssl_session_cache_hits` [456]

  The number of SSL session cache hits. This variable was added in MySQL 4.0.0.

- `Ssl_session_cache_misses` [456]

  The number of SSL session cache misses. This variable was added in MySQL 4.0.0.

- `Ssl_session_cache_mode` [456]

  The SSL session cache mode. This variable was added in MySQL 4.0.0.

- `Ssl_session_cache_overflows` [456]

  The number of SSL session cache overflows. This variable was added in MySQL 4.0.0.

- `Ssl_session_cache_size` [456]

  The SSL session cache size. This variable was added in MySQL 4.0.0.

- `Ssl_session_cache_timeouts` [456]

  The number of SSL session cache timeouts. This variable was added in MySQL 4.0.0.

- `Ssl_sessions_reused` [456]

  How many SSL connections were reused from the cache. This variable was added in MySQL 4.0.0.

- `Ssl_used_session_cache_entries` [456]

  How many SSL session cache entries were used. This variable was added in MySQL 4.0.0.

- `Ssl_verify_depth` [456]

  The verification depth for replication SSL connections. This variable was added in MySQL 4.0.0.

- `Ssl_verify_mode` [456]

The verification mode for replication SSL connections. This variable was added in MySQL 4.0.0.

- `Ssl_version` [457]

  The SSL version number. This variable was added in MySQL 4.0.0.

- `Table_locks_immediate` [457]

  The number of times that a request for a table lock could be granted immediately. This variable was added in MySQL 3.23.33.

- `Table_locks_waited` [457]

  The number of times that a request for a table lock could not be granted immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication. This variable was added in MySQL 3.23.33.

- `Threads_cached` [457]

  The number of threads in the thread cache. This variable was added in MySQL 3.23.17.

- `Threads_connected` [457]

  The number of currently open connections.

- `Threads_created` [457]

  The number of threads created to handle connections. If `Threads_created` [457] is big, you may want to increase the `thread_cache_size` [431] value. The cache miss rate can be calculated as `Threads_created` [457] divided by `Connections` [450]. This variable was added in MySQL 3.23.31.

- `Threads_running` [457]

  The number of threads that are not sleeping.

- `Uptime` [457]

  The number of seconds that the server has been up.

5.1.6 Server SQL Modes

The MySQL server can operate in different SQL modes, and (as of MySQL 4.1) can apply these modes differentially for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

Modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting `mysqld` with the `--sql-mode="modes"` [394] option, or by using `sql-mode="modes"` [394] in `my.cnf` (Unix operating systems) or `my.ini` (Windows). *`modes`* is a list of different modes separated by comma (“`,`”) characters. The default value is empty (no modes set). The *`modes`* value also can be empty (`--sql-mode=""` [394] on the command line, or `sql-mode=""` [394] in `my.cnf` on Unix systems or in `my.ini` on Windows) if you want to clear it explicitly.

Beginning with MySQL 4.1, you can change the SQL mode at runtime by using a `SET [GLOBAL|SESSION] sql_mode='modes'` statement to set the `sql_mode` [429] system value. Setting the

GLOBAL variable requires the SUPER [493] privilege and affects the operation of all clients that connect from that time on. Setting the SESSION variable affects only the current client. Any client can change its own session sql_mode [429] value at any time.

You can retrieve the current global or session sql_mode [429] value with the following statements:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

This mode changes syntax and behavior to conform more closely to standard SQL, and is available beginning in MySQL 4.1.1.

The following list describes all supported modes:

- ANSI_QUOTES [458]

  Treat "!" as an identifier quote character (like the "`" quote character) and not as a string quote character. You can still use "`" to quote identifiers with this mode enabled. With ANSI_QUOTES [458] enabled, you cannot use double quotation marks to quote literal strings, because it is interpreted as an identifier. (Added in MySQL 4.0.0)

- IGNORE_SPACE [458]

  Permit spaces between a function name and the "(" character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in Section 8.2, "Database, Table, Index, Column, and Alias Names". For example, because there is a COUNT() [882] function, the use of count as a table name in the following statement causes an error:

  ```
  mysql> CREATE TABLE count (i INT);
  ERROR 1064 (42000): You have an error in your SQL syntax
  ```

  The table name should be quoted:

  ```
  mysql> CREATE TABLE `count` (i INT);
  Query OK, 0 rows affected (0.00 sec)
  ```

  The IGNORE_SPACE [458] SQL mode applies to built-in functions, not to user-defined functions. It is always permissible to have spaces after a UDF name, regardless of whether IGNORE_SPACE [458] is enabled.

  For further discussion of IGNORE_SPACE [458], see Section 8.2.3, "Function Name Parsing and Resolution".

  (Added in MySQL 4.0.0)

- NO_AUTO_VALUE_ON_ZERO [458]

  NO_AUTO_VALUE_ON_ZERO [458] affects handling of AUTO_INCREMENT columns. Normally, you generate the next sequence number for the column by inserting either NULL or 0 into it. NO_AUTO_VALUE_ON_ZERO [458] suppresses this behavior for 0 so that only NULL generates the next sequence number. (Added in MySQL 4.1.1)

  This mode can be useful if 0 has been stored in a table's AUTO_INCREMENT column. (Storing 0 is not a recommended practice, by the way.) For example, if you dump the table with mysqldump and then reload it, MySQL normally generates new sequence numbers when it encounters the

`0` values, resulting in a table with contents different from the one that was dumped. Enabling `NO_AUTO_VALUE_ON_ZERO` [458] before reloading the dump file solves this problem. As of MySQL 4.1.1, `mysqldump` automatically includes a statement in the dump output that enables `NO_AUTO_VALUE_ON_ZERO` [458] to avoid this problem.

- `NO_DIR_IN_CREATE` [459]

  When creating a table, ignore all `INDEX DIRECTORY` and `DATA DIRECTORY` directives. This option is useful on slave replication servers. (Added in MySQL 4.0.15)

- `NO_FIELD_OPTIONS` [459]

  Do not print MySQL-specific column options in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode. (Added in MySQL 4.1.1)

- `NO_KEY_OPTIONS` [459]

  Do not print MySQL-specific index options in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode. (Added in MySQL 4.1.1)

- `NO_TABLE_OPTIONS` [459]

  Do not print MySQL-specific table options (such as `ENGINE`) in the output of `SHOW CREATE TABLE`. This mode is used by `mysqldump` in portability mode. (Added in MySQL 4.1.1)

- `NO_UNSIGNED_SUBTRACTION` [459]

  By default, subtraction between integer operands produces an `UNSIGNED` result if any operand is`UNSIGNED`. When `NO_UNSIGNED_SUBTRACTION` [459] is enabled, the subtraction result is signed, *even if any operand is unsigned*. For example, compare the type of column `c2` in table `t1` with that of column `c2` in table `t2`:

```
mysql> SET sql_mode='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| c2    | bigint(21) unsigned |      |     | 0       |       |
+-------+---------------------+------+-----+---------+-------+

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| c2    | bigint(21) |      |     | 0       |       |
+-------+------------+------+-----+---------+-------+
```

Note that this means that `BIGINT UNSIGNED` is not 100% usable in all contexts. See Section 11.10, “Cast Functions and Operators”. (Added in MySQL 4.0.2)

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|    18446744073709551615 |
```

```
+------------------------+

mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+------------------------+
|                     -1 |
+------------------------+
```

- `ONLY_FULL_GROUP_BY` [460]

  Do not permit queries for which the `SELECT` list refers to nonaggregated columns that are not named in the `GROUP BY` clause. (Added in MySQL 4.0.0) The following query is invalid with this mode enabled because `address` is not named in the `GROUP BY` clause:

```
SELECT name, address, MAX(age) FROM t GROUP BY name;
```

- `PIPES_AS_CONCAT` [460]

  Treat `||` [787] as a string concatenation operator (same as `CONCAT()` [795]) rather than as a synonym for `OR` [787]. (Added in MySQL 4.0.0)

- `REAL_AS_FLOAT` [460]

  Treat `REAL` as a synonym for `FLOAT`. By default, MySQL treats `REAL` as a synonym for `DOUBLE`. (Added in MySQL 4.0.0)

The following special modes are provided as shorthand for combinations of mode values from the preceding list. All are available as of MySQL 4.1.1.

The descriptions include all mode values that are available in the most recent version of MySQL. For older versions, a combination mode does not include individual mode values that are not available except in newer versions.

- `ANSI` [460]

  Equivalent to `REAL_AS_FLOAT` [460], `PIPES_AS_CONCAT` [460], `ANSI_QUOTES` [458], `IGNORE_SPACE` [458]. Before MySQL 4.1.11, `ANSI` [460] also includes `ONLY_FULL_GROUP_BY` [460]. See Section 1.9.3, “Running MySQL in ANSI Mode”.

- `DB2` [460]

  Equivalent to `PIPES_AS_CONCAT` [460], `ANSI_QUOTES` [458], `IGNORE_SPACE` [458], `NO_KEY_OPTIONS` [459], `NO_TABLE_OPTIONS` [459], `NO_FIELD_OPTIONS` [459].

- `MAXDB` [460]

  Equivalent to `PIPES_AS_CONCAT` [460], `ANSI_QUOTES` [458], `IGNORE_SPACE` [458], `NO_KEY_OPTIONS` [459], `NO_TABLE_OPTIONS` [459], `NO_FIELD_OPTIONS` [459].

- `MSSQL` [460]

  Equivalent to `PIPES_AS_CONCAT` [460], `ANSI_QUOTES` [458], `IGNORE_SPACE` [458], `NO_KEY_OPTIONS` [459], `NO_TABLE_OPTIONS` [459], `NO_FIELD_OPTIONS` [459].

- `MYSQL323` [460]

  Equivalent to `NO_FIELD_OPTIONS` [459].

- `MYSQL40` [461]

  Equivalent to `NO_FIELD_OPTIONS` [459].

- `ORACLE` [461]

  Equivalent to `PIPES_AS_CONCAT` [460], `ANSI_QUOTES` [458], `IGNORE_SPACE` [458], `NO_KEY_OPTIONS` [459], `NO_TABLE_OPTIONS` [459], `NO_FIELD_OPTIONS` [459].

- `POSTGRESQL` [461]

  Equivalent to `PIPES_AS_CONCAT` [460], `ANSI_QUOTES` [458], `IGNORE_SPACE` [458], `NO_KEY_OPTIONS` [459], `NO_TABLE_OPTIONS` [459], `NO_FIELD_OPTIONS` [459].

5.1.7 Server-Side Help

As of MySQL 4.1, MySQL Server supports a `HELP` statement that returns online information from the MySQL Reference manual (see Section 12.7.3, “`HELP` Syntax”). The proper operation of this statement requires that the help tables in the `mysql` database be initialized with help topic information, which is done by processing the contents of the `fill_help_tables.sql` script.

If you install MySQL using a binary or source distribution on Unix, help table setup occurs when you run `mysql_install_db`. For an RPM distribution on Linux or binary distribution on Windows, help table setup occurs as part of the MySQL installation process.

If you upgrade MySQL using a binary distribution, the help tables are not upgraded automatically, but you can upgrade them manually. Locate the `fill_help_tables.sql` file in the `share` or `share/mysql` directory. Change location into that directory and process the file with the `mysql` client as follows:

```
shell> mysql -u root mysql < fill_help_tables.sql
```

You can also obtain the latest `fill_help_tables.sql` at any time to upgrade your help tables. Download the proper file for your version of MySQL from http://dev.mysql.com/doc/index-other.html. After downloading and uncompressing the file, process it with `mysql` as described previously.

If you are working with Bazaar and a MySQL development source tree, you will need to download the `fill_help_tables.sql` file because the tree contains only a “stub” version.

5.1.8 Server Response to Signals

On Unix, signals can be sent to processes. `mysqld` responds to signals sent to it as follows:

- `SIGTERM` causes the server to shut down.

- `SIGHUP` causes the server to reload the grant tables and flush the logs (like `FLUSH PRIVILEGES` and `FLUSH LOGS`). It also writes a status report to the error log that has this format:

```
Status information:

Current dir: /var/mysql/data/
Running threads: 0  Stack size: 196608
Current locks:

Key caches:
default
Buffer_size:       8388600
Block_size:           1024
```

```
Division_limit:        100
Age_limit:             300
blocks used:             0
not flushed:             0
w_requests:              0
writes:                  0
r_requests:              0
reads:                   0

handler status:
read_key:            0
read_next:           0
read_rnd             0
read_first:          1
write:               0
delete               0
update:              0

Table status:
Opened tables:          5
Open tables:            0
Open files:             7
Open streams:           0

Alarm status:
Active alarms:   1
Max used alarms: 2
Next alarm time: 67
```

On some Mac OS X 10.3 versions, `mysqld` ignores `SIGHUP` and `SIGQUIT`.

5.1.9 The Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

   Server shutdown can be initiated several ways. For example, a user with the `SHUTDOWN` [492] privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to. (On Windows, a user with Administrator rights can also shut down the server using `NET STOP` *`service_name`*, where *`service_name`* is the name of the MySQL service. By default, this is `MySQL`.)

2. The server creates a shutdown thread if necessary.

   Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

   ```
   Error: Can't create thread to kill server
   ```

3. The server stops accepting new connections.

   To prevent new activity from being initiated during shutdown, the server stops accepting new client connections. It does this by closing the network connections to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

   For each thread that is associated with a client connection, the connection to the client is broken and the thread is marked as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see Section 12.4.6.3, “KILL Syntax”, in particular for the instructions about killed REPAIR TABLE or OPTIMIZE TABLE operations on MyISAM tables.

   For threads that have an open transaction, the transaction is rolled back. Note that if a thread is updating a nontransactional table, an operation such as a multiple-row UPDATE or INSERT may leave the table partially updated, because the operation can terminate before completion.

   If the server is a master replication server, threads associated with currently connected slaves are treated like other client threads. That is, each one is marked as killed and exits when it next checks its state.

   If the server is a slave replication server, the I/O and SQL threads, if active, are stopped before client threads are marked as killed. The SQL thread is permitted to finish its current statement (to avoid causing replication problems), and then stops. If the SQL thread was in the middle of a transaction at this point, the transaction is rolled back.

5. Storage engines are shut down or closed.

   At this stage, the table cache is flushed and all open tables are closed.

   Each storage engine performs any actions necessary for tables that it manages. For example, MyISAM flushes any pending index writes for a table. InnoDB flushes its buffer pool to disk, writes the current LSN to the tablespace, and terminates its own internal threads.

6. The server exits.

5.2 The mysqld-max Extended MySQL Server

A MySQL-Max server is a version of the mysqld MySQL server that has been built to include additional features. The MySQL-Max distribution to use depends on your platform:

- For Windows, MySQL binary distributions include both the standard server (mysqld.exe) and the MySQL-Max server (mysqld-max.exe), so no special distribution is needed. Just use a regular Windows distribution. See Section 2.3, “Installing MySQL on Microsoft Windows”.
- For Linux, if you install MySQL using RPM distributions, the MySQL-Max RPM presupposes that you have already installed the regular server RPM. Use the regular MySQL-server RPM first to install a standard server named mysqld, and then use the MySQL-Max RPM to install a server named mysqld-max. See Section 2.4, “Installing MySQL from RPM Packages on Linux”, for more information on the Linux RPM packages.
- All other MySQL-Max distributions contain a single server that is named mysqld but that has the additional features included.

You can find the MySQL-Max binaries on the MySQL Web site at http://dev.mysql.com/doc/.

We build the MySQL-Max servers by using the following configure options:

- --with-server-suffix=-max

  This option adds a -max suffix to the mysqld version string.

- `--with-innodb`

  This option enables support for the `InnoDB` storage engine. MySQL-Max servers always include `InnoDB` support, but this option actually is needed only for MySQL 3.23. From MySQL 4.0 onward, `InnoDB` is included by default in all binary distributions, so a MySQL-Max server is not needed to obtain `InnoDB` support.

- `--with-bdb`

  This option enables support for the Berkeley DB (`BDB`) storage engine on those platforms for which `BDB` is available. (See notes in the following discussion.)

- `--with-blackhole-storage-engine`

  This option enables support for the `BLACKHOLE` storage engine in MySQL 4.1.11 and newer.

- `--with-example-storage-engine`

  This option enables support for the `EXAMPLE` storage engine in MySQL 4.1.10 and newer.

- `--with-ndbcluster`

  As of MySQL 4.1.2, this option enables support for the `NDBCLUSTER` storage engine on those platforms for which Cluster is available. (See notes in the following discussion.)

- `USE_SYMDIR`

  This define is enabled to turn on database symbolic link support for Windows. This applies only before MySQL 4.0. From MySQL 4.0 onward, symbolic link support is enabled for all Windows servers, so a MySQL-Max server is not needed to take advantage of this feature.

MySQL-Max binary distributions are a convenience for those who wish to install precompiled programs. If you build MySQL using a source distribution, you can build your own Max-like server by enabling the same features at configuration time that the MySQL-Max binary distributions are built with.

MySQL-Max servers include the BerkeleyDB (`BDB`) storage engine whenever possible, but not all platforms support `BDB`.

The following table shows on which platforms MySQL-Max binaries include support for `BDB` and `NDB Cluster`:

As of MySQL 4.1.2, MySQL Cluster is supported on Linux (on most platforms), Solaris, Mac OS X, and HP-UX only. Some users have reported success in using MySQL Cluster built from source on BSD operating systems, but these are not officially supported at this time. Note that, even for servers compiled with Cluster support, the `NDBCLUSTER` storage engine is not enabled by default. You must start the server with the `--ndbcluster` [1301] option to use it as part of a MySQL Cluster. (For details, see Section 15.3, “MySQL Cluster Configuration”.)

The following table shows the platforms for which MySQL-Max binaries include support for `BDB` and `NDBCLUSTER`.

System	BDB Support	NDB Support
AIX 5.2	N	N
HP-UX	Y	Y
Linux-Alpha	N	N

System	BDB Support	NDB Support
Linux-IA-64	N	Y
Linux-Intel	Y	Y
Mac OS X	N	Y
NetWare	N	N
SCO 6	N	N
Solaris-SPARC	Y	Y
Solaris-Intel	N	Y
Solaris-AMD 64	Y	Y
Windows NT/2000/XP	Y	N

To find out which storage engines your server supports, use the `SHOW ENGINES` statement. (See Section 12.4.5.10, "`SHOW ENGINES` Syntax".) For example:

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
 Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
*************************** 2. row ***************************
 Engine: HEAP
Support: YES
Comment: Alias for MEMORY
*************************** 3. row ***************************
 Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 4. row ***************************
 Engine: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
...
```

Before MySQL 4.1.2, `SHOW ENGINES` is unavailable. Use the following statement instead and check the value of the variable for the storage engine in which you are interested:

```
mysql> SHOW VARIABLES LIKE 'have%';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| have_archive         | YES   |
| have_bdb             | YES   |
| have_blackhole_engine | YES   |
| have_compress        | YES   |
| have_crypt           | YES   |
| have_csv             | YES   |
| have_example_engine  | YES   |
| have_geometry        | YES   |
| have_innodb          | YES   |
| have_isam            | NO    |
| have_ndbcluster      | NO    |
| have_openssl         | YES   |
| have_query_cache     | YES   |
| have_raid            | NO    |
| have_rtree_keys      | YES   |
| have_symlink         | YES   |
+----------------------+-------+
```

```
16 rows in set (0.00 sec)
```

The precise output from these statements may vary according to the MySQL version used (and the features that are enabled). The values of the second column of the output indicate the server's level of support for each feature, as shown here:

Value	Meaning
YES	The feature is supported and is active.
NO	The feature is not supported.
DISABLED	The feature is supported but has been disabled.

A value of NO means that the server was compiled without support for the feature, so it cannot be activated at runtime.

A value of DISABLED occurs either because the server was started with an option that disables the feature, or because not all options required to enable it were given. In the latter case, the error log file should contain a reason indicating why the option is disabled. See Section 5.3.1, “The Error Log”.

One situation in which you might see DISABLED occurs with MySQL 3.23 when the InnoDB storage engine is compiled in. In MySQL 3.23, you must supply at least the innodb_data_file_path [1067] option at runtime to set up the InnoDB tablespace. Without this option, InnoDB disables itself. See Section 13.2.2, “InnoDB in MySQL 3.23”. You can specify configuration options for the BDB storage engine, too, but BDB does not disable itself if you do not provide them. See Section 13.5.3, “BDB Startup Options”.

You might also see DISABLED for a storage engine if the server was compiled to support it, but was started with a --skip-*engine_name* option. For example, --skip-innodb [1066] disables the InnoDB engine. For the NDB Cluster storage engine, DISABLED means the server was compiled with support for MySQL Cluster, but was not started with the --ndb-cluster option.

As of version 3.23, all MySQL servers support MyISAM tables, because MyISAM is the default storage engine.

5.3 MySQL Server Logs

MySQL Server has several logs that can help you find out what activity is taking place.

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping mysqld
ISAM log	Changes to the ISAM tables (used only for debugging the ISAM code)
General query log	Established client connections and statements received from clients
Update log	Statements that change data (this log is deprecated)
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication master server
Slow query log	Queries that took more than long_query_time [417] seconds to execute

By default, all log files are created in the data directory. You can force the server to close and reopen the log files (or in some cases switch to a new log file) by flushing the logs. Log flushing occurs when you issue a FLUSH LOGS statement or execute a mysqladmin flush-logs, mysqladmin refresh,

`mysqldump --flush-logs`, or `mysqldump --master-data` command. See Section 12.4.6.2, "`FLUSH` Syntax", Section 4.5.2, "`mysqladmin` — Client for Administering a MySQL Server", and Section 4.5.4, "`mysqldump` — A Database Backup Program". In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` [1184] system variable.

The relay log is used only on slave replication servers, to hold data changes from the master server that must also be made on the slave. For discussion of relay log contents and configuration, see Section 14.3.2, "The Slave Relay Log".

For information about log maintenance operations such as expiration of old log files, see Section 5.3.6, "Server Log Maintenance".

See Section 5.4.2.1, "Administrator Guidelines for Password Security", for information about keeping logs secure.

5.3.1 The Error Log

The error log contains information indicating when `mysqld` was started and stopped and also any critical errors that occur while the server is running. If `mysqld` notices a table that needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if `mysqld` dies. The trace can be used to determine where `mysqld` died. See Section 18.4, "Porting to Other Systems".

Beginning with MySQL 4.0.10, you can specify where `mysqld` writes the error log with the `--log-error[=file_name]` [388] option. If the option is given with no *`file_name`* value, `mysqld` uses the name *`host_name`*`.err` by default. The server creates the file in the data directory unless an absolute path name is given to specify a different directory. (Prior to MySQL 4.0.10, the Windows error log name is `mysql.err`.) If you flush the logs using `FLUSH LOGS` or `mysqladmin flush-logs`, `mysqld` renames the current log file with the suffix `-old`, then creates a new empty log file. Be aware that a second log-flushing operation thus causes the original error log file to be lost unless you save it under a different name. For example, you can use the following commands to save the file:

```
shell> mysqladmin flush-logs
shell> mv host_name.err-old backup-directory
```

No error log renaming occurs when the logs are flushed if the server is not writing to a named file.

In older MySQL versions on Unix, error log handling was done by `mysqld_safe` which redirected the error file to *`host_name`*`.err`. You could change this file name by specifying a `--err-log=`*`file_name`* [244] option to `mysqld_safe`.

If you do not specify `--log-error` [388], or (on Windows) if you use the `--console` [385] option, errors are written to `stderr`, the standard error output. Usually this is your terminal.

On Windows, error output is always written to the `.err` file if `--console` [385] is not given.

In addition, on Windows, events and error messages are written to the Windows Event Log within the Application log. Entries marked as `Warning` and `Note` are written to the Event Log, but informational messages (such as information statements from individual storage engines) are not copied to the Event Log. The log entries have a source of **MySQL**. You cannot disable writing information to the Windows Event Log.

The `--log-warnings` [389] option or `log_warnings` [417] system variable can be used to control warning logging to the error log. The default value is enabled (1) as of MySQL 4.0.19 and 4.1.2. Warning logging can be disabled using a value of 0. As of MySQL 4.0.21 and 4.1.3, the value can be greater than

1. If the value is greater than 1, aborted connections are written to the error log. See Section B.5.2.11, “Communication Errors and Aborted Connections”.

If `mysqld_safe` is used to start `mysqld` and `mysqld` dies unexpectedly, `mysqld_safe` notices that it needs to restart `mysqld` and writes a `restarted mysqld` message to the error log.

5.3.2 The General Query Log

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`.

Older versions of the `mysql.server` script (from MySQL 3.23.4 to 3.23.8) pass a `--log` [388] option to `safe_mysqld` to enable the general query log. If you need better performance when you start using MySQL in a production environment, you can remove the `--log` [388] option from `mysql.server` or change it to `--log-bin` [1181]. See Section 5.3.4, “The Binary Log”.

`mysqld` writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order contrasts to the update log and the binary log, which are written after the query is executed but before any locks are released. (Also, the query log contains all statements, whereas the update and binary logs do not contain statements that only select data.)

To enable the general query log, start `mysqld` with the `--log[=file_name]` [388] or `-l [file_name]` option.

If the general query log file is enabled but no name is specified, the default name is *`host_name`*`.log` and the server creates the file in the same directory where it creates the PID file. If a name is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). On Unix, you can rename the file and create a new one by using the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> mv host_name-old.log backup-directory
```

On Windows, you cannot rename the log file while the server has it open. You must stop the server, rename the file, and then restart the server to create a new log file.

To disable or enable general query logging for the current connection, set the session `sql_log_off` [429] variable to `ON` or `OFF`.

The general query log should be protected because logged statements might contain passwords. See Section 5.4.2.1, “Administrator Guidelines for Password Security”.

5.3.3 The Update Log

Note

The update log has been deprecated and replaced by the more useful, informative, and efficient binary log. See Section 5.3.4, “The Binary Log”.

When started with the `--log-update[=file_name]` [389] option, `mysqld` writes a log file containing all SQL statements that update data. If no *`file_name`* value is given, the default name is name of the host machine. If a file name is given, but it does not contain a leading path, the file is written in the data

directory. If `file_name` does not have an extension, `mysqld` creates log files with names of the form *`file_name.nnnnnn`*, where *`nnnnnn`* is a number that is incremented each time you start the server or flush the logs.

Note

For this naming scheme to work, you must not create your own files with the same names as those that might be used in the log file sequence.

Update logging is "smart" in that *it logs only statements that actually update data*. Thus, an `UPDATE` or `DELETE` with a `WHERE` clause that finds no rows is not written to the log. Update logging also skips `UPDATE` statements that merely set a column to its existing value.

The update logging is done immediately after a query completes but before any locks are released or any commit is done. This ensures that statements are logged in execution order.

If you want to update a database from update log files, you could do the following (assuming that your update logs have names of the form `file_name.nnnnnn`):

```
shell> ls -1 -t -r file_name.[0-9]* | xargs cat | mysql
```

`ls` is used to sort the update log file names into the right order.

This can be useful if you have to revert to backup files after a crash and you want to redo the updates that occurred between the time of the backup and the crash.

The update log should be protected because logged statements might contain passwords. See Section 5.4.2.1, "Administrator Guidelines for Password Security".

5.3.4 The Binary Log

The binary log contains "events" that describe database changes such as table creation operations or changes to table data. As of MySQL 4.1.3, it also contains events for statements that potentially could have made changes (for example, a `DELETE` which matched no rows). The binary log also contains information about how long each statement took that updated data. The binary log has two important purposes:

- For replication, the binary log is used on master replication servers as a record of the statements to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See Section 14.2, "Replication Implementation Overview".
- Certain data recovery operations require use of the binary log. After a backup has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 6.5, "Point-in-Time (Incremental) Recovery Using the Binary Log".

Note

The binary log has replaced the old update log, which is no longer available as of MySQL 5.0. The binary log contains all information that is available in the update log in a more efficient format and in a manner that is transaction-safe. If you are using transactions, you must use the MySQL binary log for backups instead of the old update log.

Running a server with binary logging enabled makes performance slightly slower. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

For information about server options and variables affecting the operation of binary logging, see Section 14.8.4, “Binary Log Options and Variables”.

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. If you want to log all statements (for example, to identify a problem query), use the general query log. See Section 5.3.2, “The General Query Log”.

The binary log should be protected because logged statements might contain passwords. See Section 5.4.2.1, “Administrator Guidelines for Password Security”.

For detailed information about the format of the binary log, see MySQL Internals: The Binary Log.

To enable the binary log, start the server with the `--log-bin[=base_name]` [1181] option. If no *`base_name`* value is given, the default name is the value of the `pid-file` option (which by default is the name of host machine) followed by `-bin`. If the basename is given, the server writes the file in the data directory unless the basename is given with a leading absolute path name to specify a different directory. It is recommended that you specify a basename; see Section B.5.8.4, “Open Issues in MySQL”, for the reason.

If you supply an extension in the log name (for example, `--log-bin=base_name.extension` [1181]), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log basename to generate binary log file names. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new file in the series each time it starts or flushes the logs. The server also creates a new binary log file automatically after the current log's size reaches `max_binlog_size` [1184]. A binary log file may become larger than `max_binlog_size` [1184] if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, `mysqld` also creates a binary log index file that contains the names of all used binary log files. By default, this has the same basename as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=file_name]` [1182] option. You should not manually edit this file while `mysqld` is running; doing so would confuse `mysqld`.

The term “binary log file” generally denotes an individual numbered file containing database events. The term “binary log” collectively denotes the set of numbered binary log files plus the index file.

The server evaluates the `--binlog-do-db` [1182] and `--binlog-ignore-db` [1182] options in the same way as it does the `--replicate-do-db` [1176] and `--replicate-ignore-db` [1176] options. For information about how this is done, see Section 14.9.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

A replication slave server by default does not write to its own binary log any data modifications that are received from the replication master. To log these modifications, start the slave with the `--log-slave-updates` [1173] option in addition to the `--log-bin` [1181] option (see Section 14.8.3, “Replication Slave Options and Variables”). This is done when a slave is also to act as a master to other slaves in chained replication.

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE BINARY LOGS`. See Section 12.4.6.5, “`RESET` Syntax”, and Section 12.5.1.1, “`PURGE BINARY LOGS` Syntax”.

If you are using replication, you should not delete old binary log files on the master until you are sure that no slave still needs to use them. For example, if your slaves never run more than three days behind, once a day you can execute `mysqladmin flush-logs` on the master and then remove any logs that are

more than three days old. You can remove the files manually, but it is preferable to use PURGE BINARY LOGS, which also safely updates the binary log index file for you (and which can take a date argument as of MySQL 4.1). See Section 12.5.1.1, “PURGE BINARY LOGS Syntax”.

A client that has the SUPER [493] privilege can disable binary logging of its own statements by using a SET sql_log_bin=0 statement. See Section 5.1.3, “Server System Variables”.

You can display the contents of binary log files with the mysqlbinlog utility. This can be useful when you want to reprocess statements in the log for a recovery operation. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

mysqlbinlog also can be used to display replication slave relay log file contents because they are written using the same format as binary log files. For more information on the mysqlbinlog utility and how to use it, see Section 4.6.6, “mysqlbinlog — Utility for Processing Binary Log Files”. For more information about the binary log and recovery operations, see Section 6.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Binary logging is done immediately after a statement completes but before any locks are released or any commit is done. This ensures that the log is logged in execution order.

Updates to nontransactional tables are stored in the binary log immediately after execution. Within an uncommitted transaction, all updates (UPDATE, DELETE, or INSERT) that change transactional tables such as BDB or InnoDB tables are cached until a COMMIT statement is received by the server. At that point, mysqld writes the entire transaction to the binary log before the COMMIT is executed.

Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a ROLLBACK statement at the end to ensure that the modifications to those tables are replicated. This is true as of MySQL 4.0.15.

When a thread that handles the transaction starts, it allocates a buffer of binlog_cache_size [407] to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

The Binlog_cache_use [449] status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The Binlog_cache_disk_use [449] status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning binlog_cache_size [407] to a large enough value that avoids the use of temporary files.

The max_binlog_cache_size [1183] system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the update log or binary log, concurrent inserts are converted to normal inserts for CREATE ... SELECT or INSERT ... SELECT statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation.

The binary log format has some known limitations that can affect recovery from backups, especially in old versions. These caveats, which also affect replication, are listed at Section 14.7, “Replication Features and Issues”. One caveat which does not affect replication but only recovery with mysqlbinlog: before MySQL 4.1, mysqlbinlog could not prepare output suitable for mysql if the binary log contained interlaced statements originating from different clients that used temporary tables of the same name. This is fixed in MySQL 4.1. However, the problem still existed for LOAD DATA INFILE statements until it was fixed in MySQL 4.1.8.

The binary log format differs between versions 3.23 and 4.0. (These format changes were required to implement enhancements to replication.) However, MySQL 4.1 has the same binary log format as 4.0. See Section 14.5, “Replication Compatibility Between MySQL Versions”.

Before MySQL 4.1.9, writes to a binary log file or binary log index file that failed due to a full disk or an exceeded quota resulted in corruption of the file. Starting from MySQL 4.1.9, writes to the binary log file and binary log index file are handled the same way as writes to `MyISAM` tables. See Section B.5.4.3, “How MySQL Handles a Full Disk”.

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes, there is a chance that the last statements of the binary log are lost. To prevent this, you can make the binary log be synchronized to disk after every *`N`* writes to the binary log, with the `sync_binlog` [1184] system variable. See Section 5.1.3, “Server System Variables”. 1 is the safest value for `sync_binlog` [1184], but also the slowest. Even with `sync_binlog` [1184] set to 1, there is still the chance of an inconsistency between the table content and binary log content in case of a crash. For example, if you are using `InnoDB` tables and the MySQL server processes a `COMMIT` statement, it writes the whole transaction to the binary log and then commits this transaction into `InnoDB`. If the server crashes between those two operations, the transaction is rolled back by `InnoDB` at restart but still exists in the binary log. This problem can be solved with the `--innodb-safe-binlog` [387] option (available starting from MySQL 4.1.3), which adds consistency between the content of `InnoDB` tables and the binary log.

For this option to provide a greater degree of safety, the MySQL server should also be configured to synchronize the binary log and the `InnoDB` logs to disk at every transaction. The `InnoDB` logs are synchronized by default, and `sync_binlog=1` can be used to synchronize the binary log. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server cuts rolled back `InnoDB` transactions from the binary log. This ensures that the binary log reflects the exact data of `InnoDB` tables, and so, that the slave remains in synchrony with the master (not receiving a statement which has been rolled back).

Note that `--innodb-safe-binlog` [387] can be used even if the MySQL server updates other storage engines than `InnoDB`. Only statements and transactions that affect `InnoDB` tables are subject to removal from the binary log at `InnoDB`'s crash recovery. If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed `InnoDB` transaction. This should not happen if `sync_binlog=1` and the disk/file system do an actual sync when they are requested to (some do not), so the server prints an error message `The binary log file_name is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the master's data.

5.3.5 The Slow Query Log

The slow query log consists of SQL statements that took more than `long_query_time` [417] seconds to execute. The minimum and default values of `long_query_time` [417] are 1 and 10, respectively.

The time to acquire the initial table locks is not counted as execution time. `mysqld` writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might differ from execution order.

To enable the slow query log, start `mysqld` with the `--log-slow-queries[=`*`file_name`*`]` [388] option.

If the slow query log file is enabled but no name is specified, the default name is *`host_name`*`-slow.log` and the server creates the file in the same directory where it creates the PID file. If a name is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory.

Before MySQL 4.1, if you also use `--log-long-format` [388] when logging slow queries, queries that are not using indexes are logged as well. Starting with MySQL 4.1, logging of queries not using indexes for row lookups is enabled using the `--log-queries-not-using-indexes` [388] option instead. The `--log-long-format` [388] is deprecated as of MySQL 4.1, when `--log-short-format` [388] was introduced, which causes less information to be logged. (The long log format is the default setting since version 4.1.) See Section 5.1.2, “Server Command Options”.

In MySQL 4.0, slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` were written to the slow query log. This logging was disabled in MySQL 4.1 until 4.1.13, when the `--log-slow-admin-statements` [388] server option was added to specify logging of slow administrative statements.

The server does not write queries handled by the query cache to the slow query log, nor queries that would not benefit from the presence of an index because the table has zero rows or one row.

Replication slaves do not write replicated queries to the slow query log, even if the same queries were written to the slow query log on the master. This is a known issue. (Bug #23300)

The slow query log should be protected because logged statements might contain passwords. See Section 5.4.2.1, “Administrator Guidelines for Password Security”.

The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can become a difficult task. To make this easier, you can process a slow query log file using the `mysqldumpslow` command to summarize the queries that appear in the log. See Section 4.6.7, “`mysqldumpslow` — Summarize Slow Query Log Files”.

5.3.6 Server Log Maintenance

As described in Section 5.3, “MySQL Server Logs”, MySQL Server can create several different log files to help you see what activity is taking place. However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See Section 6.2, “Database Backup Methods”.

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for this. If you installed MySQL from an RPM distribution, this script should have been installed automatically. Be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all slaves.

On other systems, you must install a short script yourself that you start from `cron` (or its equivalent) for handling log files.

For the binary log, you can set the `expire_logs_days` [411] system variable to expire binary log files automatically after a given number of days (see Section 5.1.3, “Server System Variables”). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master. To remove binary logs on demand, use the `PURGE BINARY LOGS` statement (see Section 12.5.1.1, “`PURGE BINARY LOGS` Syntax”).

You can force MySQL to start using new log files by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement or execute a `mysqladmin flush-logs`, `mysqladmin refresh`, `mysqldump --flush-logs`, or `mysqldump --master-data` command. See Section 12.4.6.2, “`FLUSH` Syntax”, Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` [1184] system variable.

A log-flushing operation does the following:

- If general query logging (`--log` [388]) or slow query logging (`--log-slow-queries` [388]) to a log file is enabled, the server closes and reopens the general query log file or slow query log file.
- If update logging (`--log-update` [389]) or binary logging (`--log-bin` [1181]) is used, the server closes the log and opens a new log file with a higher sequence number.
- If the server was started with the `--log-error` [388] option to cause the error log to be written to a file, it renames the current log file with the suffix `-old` and creates a new empty error log file.

The server creates a new binary log file when you flush the logs. However, it just closes and reopens the general and slow query log files. To cause new files to be created on Unix, rename the current log files before flushing them. At flush time, the server opens new log files with the original names. For example, if the general and slow query log files are named `mysql.log` and `mysql-slow.log`, you can use a series of commands like this:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

On Windows, use `rename` rather than `mv`.

At this point, you can make a backup of `mysql.old` and `mysql-slow.old` and then remove them from disk.

On Windows, you cannot rename log files while the server has them open. You must stop the server and rename them, and then restart the server to create new logs. For the error log, you can rename the file without a restart as described in Section 5.3.1, “The Error Log”.

5.4 General Security Issues

This section describes some general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see Section 5.5, “The MySQL Access Privilege System”.

5.4.1 General Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, we emphasize the necessity of fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines whenever possible:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` database!** This is critical, particularly before MySQL 4.1, when *the encrypted password is the real*

password in MySQL: Anyone who knows the password that is listed in the `mysql.user` table and who has access to the host listed for the account *can easily log in as that user*. In MySQL 4.1, the password hashing algorithm was changed so that this is no longer true.

- Learn the MySQL access privilege system. The `GRANT` and `REVOKE` statements are used for controlling access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

  Checklist:

  - Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL `root` user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See Section 2.10.3, “Securing the Initial MySQL Accounts”.

  - Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.

- Do not store any plaintext passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `MD5()` [868], `SHA1()` [869], or some other one-way hashing function and store the hash value.

- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Mary had a little lamb” results in a password of “Mhall”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

  Checklist:

  - Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. Another simple way to check whether or not your MySQL port is open is to try the following command from some remote machine, where *`server_host`* is the host name or IP address of the host on which your MySQL server runs:

    ```
    shell> telnet server_host 3306
    ```

    If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open. If `telnet` hangs or the connection is refused, the port is blocked, which is how you want it to be.

- Do not trust any data entered by users of your applications. They can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like “`; DROP DATABASE mysql;`”. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

  A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value `234`, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotation marks around the numeric constants: `SELECT * FROM table`

`WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing nonnumeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is permissible to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Try to enter single and double quotation marks ("`'`" and "`"`") in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
- Try to modify dynamic URLs by adding `%22` ("`"`"), `%23` ("`#`"), and `%27` ("`'`") to them.
- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.
- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!
- Check the size of data before passing it to MySQL.
- Have your application connect to the database using a user name different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

- Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:
  - MySQL C API: Use the `mysql_real_escape_string()` API call.
  - MySQL++: Use the `escape` and `quote` modifiers for query streams.
  - PHP: Use the `mysql_real_escape_string()` function (available as of PHP 4.3.0, prior to that PHP version use `mysql_escape_string()`, and prior to PHP 4.0.3, use `addslashes()`). Note that only `mysql_real_escape_string()` is character set-aware; the other functions can be "bypassed" when using (invalid) multi-byte character sets. In PHP 5 (and as of MySQL 4.1), you can use the `mysqli` extension, which supports the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders.
  - Perl DBI: Use placeholders or the `quote()` method.
  - Ruby DBI: Use placeholders or the `quote()` method.
  - Java JDBC: Use a `PreparedStatement` object and placeholders.

  Other programming interfaces might have similar capabilities.

- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections as of version 4.0. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.

- Learn to use the `tcpdump` and `strings` utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

This works under Linux and should work with small modifications under other systems.

Warning

If you do not see plaintext data, this does not always mean that the information actually is encrypted. If you need high security, you should consult with a security expert.

5.4.2 Password Security in MySQL

Passwords occur in several contexts within MySQL. The following sections provide guidelines that enable administrators and end users to keep these passwords secure and avoid exposing them. There is also a discussion of how MySQL uses password hashing internally.

5.4.2.1 Administrator Guidelines for Password Security

Database administrators should use the following guidelines to keep passwords secure.

MySQL stores passwords for user accounts in the `mysql.user` table. Access to this table should never be granted to any nonadministrative accounts. Passwords in the `user` table are stored in encrypted form, but in versions of MySQL earlier than 4.1, knowing the encrypted password for an account makes it possible to connect to the server using that account.

Passwords can appear as plain text in SQL statements such as `GRANT` and `SET PASSWORD`, or statements that invoke the `PASSWORD()` [868] function. If these statements are logged by the MySQL server, the passwords become available to anyone with access to the logs. This applies to the general query log, the slow query log, the update log, and the binary log (see Section 5.3, “MySQL Server Logs”). To guard against unwarranted exposure to log files, they should be located in a directory that restricts access to only the server and the database administrator.

Replication slaves store the password for the replication master in the `master.info` file. Access to this file should be restricted to the database adminstrator.

Database backups that include tables or log files containing passwords should be protected using a restricted access mode.

5.4.2.2 End-User Guidelines for Password Security

MySQL users should use the following guidelines to keep passwords secure.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method. In short, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

- Use a `-p`*`your_pass`* or `--password=`*`your_pass`* option on the command line. For example:

```
shell> mysql -u francis -pfrank db_name
```

This is convenient *but insecure*, because your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically

overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If your operating environment is set up to display your current command in the title bar of your terminal window, the password remains visible as long as the command is running, even if the command has scrolled out of view in the window content area.

- Use the `-p` or `--password` option on the command line with no password value specified. In this case, the client program solicits the password interactively:

```
shell> mysql -u francis -p db_name
Enter password: ********
```

The "`*`" characters indicate where you enter your password. The password is not displayed as you enter it.

It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs noninteractively, there is no opportunity to enter the password from the keyboard. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

```
[client]
password=your_pass
```

To keep the password safe, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

```
shell> chmod 600 .my.cnf
```

To name from the command line a specific option file containing the password, use the `--defaults-file=file_name` [235] option, where `file_name` is the full path name to the file. For example:

```
shell> mysql --defaults-file=/home/francis/mysql-opts
```

Section 4.2.3.3, "Using Option Files", discusses option files in more detail.

- Store your password in the `MYSQL_PWD` environment variable. See Section 2.13, "Environment Variables".

This method of specifying your MySQL password must be considered *extremely insecure* and should not be used. Some versions of `ps` include an option to display the environment of running processes. If you set `MYSQL_PWD`, your password is exposed to any other user who runs `ps`. Even on systems without such a version of `ps`, it is unwise to assume that there are no other methods by which users can examine process environments.

On Unix, the `mysql` client writes a record of executed statements to a history file (see Section 4.5.1.3, "`mysql` Logging"). By default, this file is named `.mysql_history` and is created in your home directory. Passwords can appear as plain text in SQL statements such as `GRANT` and `SET PASSWORD`, so if you use

these statements, they are logged in the history file. To keep this file safe, use a restrictive access mode, the same way as described earlier for the `.my.cnf` file.

If your command interpreter is configured to maintain a history, any file in which the commands are saved will contain MySQL passwords entered on the command line. For example, `bash` uses `~/.bash_history`. Any such file should have a restrictive access mode.

5.4.2.3 Password Hashing in MySQL

MySQL user accounts are listed in the `user` table of the `mysql` database. Each MySQL account is assigned a password, although what is stored in the `Password` column of the `user` table is not the plaintext version of the password, but a hash value computed from it. Password hash values are computed by the `PASSWORD()` [868] function.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account that the client wants to use.
- After the client connects, it can (if it has sufficient privileges) set or change the password hashes for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` [868] function to generate a password hash, or by using the `GRANT` or `SET PASSWORD` statements.

In other words, the server *uses* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` [868] function or uses a `GRANT` or `SET PASSWORD` statement to set or change a password.

The password hashing mechanism was updated in MySQL 4.1 to provide better security and to reduce the risk of passwords being intercepted. However, this new mechanism is understood only by the 4.1 server and 4.1 clients, which can result in some compatibility problems. A 4.1 client can connect to a pre-4.1 server, because the client understands both the old and new password hashing mechanisms. However, a pre-4.1 client that attempts to connect to a 4.1 server may run into difficulties. For example, a 4.0 `mysql` client that attempts to connect to a 4.1 server may fail with the following error message:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

Another common example of this phenomenon occurs for attempts to use the older PHP `mysql` extension after upgrading to MySQL 4.1 or newer. (See Common Problems with MySQL and PHP.)

The following discussion describes the differences between the old and new password mechanisms, and what you should do if you upgrade your server to 4.1 but need to maintain backward compatibility with pre-4.1 clients. Additional information can be found in Section B.5.2.4, “`Client does not support authentication protocol`”. This information is of particular importance to PHP programmers migrating MySQL databases from version 4.0 or lower to version 4.1 or higher.

Note

This discussion contrasts 4.1 behavior with pre-4.1 behavior, but the 4.1 behavior described here actually begins with 4.1.1. MySQL 4.1.0 is an “odd” release because it has a slightly different mechanism than that implemented in 4.1.1 and up. Differences between 4.1.0 and more recent versions are described further in Section 5.4.2.5, “Password Hashing in MySQL 4.1.0”.

Prior to MySQL 4.1, password hashes computed by the `PASSWORD()` [868] function are 16 bytes long. Such hashes look like this:

```
mysql> SELECT PASSWORD('mypass');
+-------------------+
| PASSWORD('mypass') |
+-------------------+
| 6f8c114b58f2ce9e   |
+-------------------+
```

The `Password` column of the `user` table (in which these hashes are stored) also is 16 bytes long before MySQL 4.1.

As of MySQL 4.1, the `PASSWORD()` [868] function has been modified to produce a longer 41-byte hash value:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
```

Accordingly, the `Password` column in the `user` table also must be 41 bytes long to store these values:

- If you perform a new installation of MySQL 4.1, the `Password` column is made 41 bytes long automatically.
- If you upgrade an older installation to 4.1, you should run the `mysql_fix_privilege_tables` script to increase the length of the `Password` column from 16 to 41 bytes. (The script does not change existing password values, which remain 16 bytes long.)

A widened `Password` column can store password hashes in both the old and new formats. The format of any given password hash value can be determined two ways:

- The obvious difference is the length (16 bytes versus 41 bytes).
- A second difference is that password hashes in the new format always begin with a "`*`" character, whereas passwords in the old format never do.

The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the `Password` column:

- If the column is short, only short-hash authentication is used.
- If the column is long, it can hold either short or long hashes, and the server can use either format:
  - Pre-4.1 clients can connect, although because they know only about the old hashing mechanism, they can authenticate only using accounts that have short hashes.
  - 4.1 clients can authenticate using accounts that have short or long hashes.

Even for short-hash accounts, the authentication process is actually a bit more secure for 4.1 and later clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating with short password hash

- 4.1 client authenticating with short password hash
- 4.1 client authenticating with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the `Password` column and by the `--old-passwords` [391] option. A 4.1 server generates long hashes only if certain conditions are met: The `Password` column must be wide enough to hold long values and the `--old-passwords` [391] option must not be given. These conditions apply as follows:

- The `Password` column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using `PASSWORD()` [868], `GRANT`, or `SET PASSWORD`. This is the behavior that occurs if you have upgraded to 4.1 but have not yet run the `mysql_fix_privilege_tables` script to widen the `Password` column.
- If the `Password` column is wide, it can store either short or long password hashes. In this case, `PASSWORD()` [868], `GRANT`, and `SET PASSWORD` generate long hashes unless the server was started with the `--old-passwords` [391] option. That option forces the server to generate short password hashes instead.

The purpose of the `--old-passwords` [391] option is to enable you to maintain backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option does not affect authentication (4.1 clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the `user` table as the result of a password-changing operation. Were that to occur, the account no longer could be used by pre-4.1 clients. Without the `--old-passwords` [391] option, the following undesirable scenario is possible:

- An old client connects to an account that has a short password hash.
- The client changes its own password. Without `--old-passwords` [391], this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the new hashing mechanism during authentication. (Once an account has a long password hash in the user table, only 4.1 clients can authenticate for it, because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is dangerous to run a 4.1 server without using the `--old-passwords` [391] option. By running the server with `--old-passwords` [391], password-changing operations do not generate long password hashes and thus do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of the `--old-passwords` [391] option is that any passwords you create or change use short hashes, even for 4.1 clients. Thus, you lose the additional security provided by long password hashes. If you want to create an account that has a long hash (for example, for use by 4.1 clients), you must do so while running the server without `--old-passwords` [391].

The following scenarios are possible for running a 4.1 or later server:

Scenario 1: Short `Password` column in user table:

- Only short hashes can be stored in the `Password` column.
- The server uses only short hashes during client authentication.
- For connected clients, password hash-generating operations involving `PASSWORD()` [868], `GRANT`, or `SET PASSWORD` use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

- The `--old-passwords` [391] option can be used but is superfluous because with a short `Password` column, the server generates only short password hashes anyway.

Scenario 2: Long `Password` column; server not started with `--old-passwords` [391] option:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only using accounts that have short hashes.
- For connected clients, password hash-generating operations involving `PASSWORD()` [868], `GRANT`, or `SET PASSWORD` use long hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made using `GRANT`, `PASSWORD()` [868], or `SET PASSWORD` results in the account being given a long password hash. From that point on, no pre-4.1 client can authenticate to that account until the client upgrades to 4.1.

To deal with this problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` [868] function instead:

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` [868] is useful for situations in which you explicitly want to generate a short hash.

Scenario 3: Long `Password` column; server started with `--old-passwords` [391] option:

- Short or long hashes can be stored in the `Password` column.
- 4.1 clients can authenticate for accounts that have short or long hashes (but note that it is possible to create long hashes only when the server is started without `--old-passwords` [391]).
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving `PASSWORD()` [868], `GRANT`, or `SET PASSWORD` use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, you cannot create accounts that have long password hashes, because the `--old-passwords` [391] option prevents generation of long hashes. Also, if you create an account with a long hash before using the `--old-passwords` [391] option, changing the account's password while `--old-passwords` [391] is in effect results in the account being given a short password, causing it to lose the security benefits of a longer hash.

The disadvantages for these scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using `OLD_PASSWORD()` [868].

In scenario 3, `--old-passwords` [391] prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes, and you cannot change them back to long hashes while `--old-passwords` [391] is in effect.

5.4.2.4 Implications of Password Hashing Changes in MySQL 4.1 for Application Programs

An upgrade to MySQL 4.1 can cause a compatibility issue for applications that use `PASSWORD()` [868] to generate passwords for their own purposes. Applications really should not do this, because `PASSWORD()` [868] should be used only to manage passwords for MySQL accounts. But some applications use `PASSWORD()` [868] for their own purposes anyway.

If you upgrade to 4.1 and run the server under conditions where it generates long password hashes, an application that uses `PASSWORD()` [868] for its own passwords breaks. The recommended course of action is to modify the application to use another function, such as `SHA1()` [869] or `MD5()` [868], to produce hashed values. If that is not possible, you can use the `OLD_PASSWORD()` [868] function, which is provided to generate short hashes in the old format. But note that `OLD_PASSWORD()` [868] may one day no longer be supported.

If the server is running under circumstances where it generates short hashes, `OLD_PASSWORD()` [868] is available but is equivalent to `PASSWORD()` [868].

PHP programmers migrating their MySQL databases from version 4.0 or lower to version 4.1 or higher should see MySQL and PHP.

5.4.2.5 Password Hashing in MySQL 4.1.0

Password hashing in MySQL 4.1.0 differs from hashing in 4.1.1 and up. The 4.1.0 differences are:

- Password hashes are 45 bytes long rather than 41 bytes.
- The `PASSWORD()` [868] function is nonrepeatable. That is, with a given argument *`X`*, successive calls to `PASSWORD(`*`X`*`)` [868] generate different results.

These differences make authentication in 4.1.0 incompatible with that of releases that follow it. If you have upgraded to MySQL 4.1.0, it is recommended that you upgrade to a newer version as soon as possible. After you do, reassign any long passwords in the `user` table so that they are compatible with the 41-byte format.

5.4.3 Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted in clear text over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are still using pre-4.1.1-style passwords, the encryption algorithm is not as strong as the newer algorithm. With some effort, a clever attacker who can sniff the traffic between the client and the server can crack the password. (See Section 5.4.2.3, “Password Hashing in MySQL”, for a discussion of the different password handling methods.)

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol (in MySQL 3.22 and above) to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure in MySQL 4.0 and up. See Section 5.6.6, “Using SSL for Secure Connections”. Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at http://www.openssh.org/, and a commercial SSH client at http://www.ssh.com/.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any user name to the client program. For example, anyone can use the `mysql` program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if `other_user` has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

  For a discussion of methods for setting passwords, see Section 5.6.5, “Assigning Account Passwords”.

- Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running `mysqld`.

- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` [491] privilege is able to cause the server to create files as `root` (for example, `~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` [395] option.

  `mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the user name in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

  ```
  [mysqld]
  user=mysql
  ```

  This causes the server to start as the designated user whether you start it manually or by using `mysqld_safe` or `mysql.server`. For more details, see Section 5.4.6, “How to Run MySQL as a Normal User”.

  Running `mysqld` as a Unix user other than `root` does not mean that you need to change the `root` user name in the `user` table. *User names for MySQL accounts have nothing to do with user names for Unix accounts*.

- Do not grant the `PROCESS` [492] or `SUPER` [493] privilege to nonadministrative users. The output of `mysqladmin processlist` and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is permitted to see the server process list might be able to see statements issued by other users such as `UPDATE user SET password=PASSWORD('not_secure')`.

  `mysqld` reserves an extra connection for users who have the `SUPER` [493] privilege (`PROCESS` [492] before MySQL 4.0.2), so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

  The `SUPER` [493] privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not grant the `FILE` [491] privilege to nonadministrative users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the `mysqld` daemon. To make this a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

  The `FILE` [491] privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

- Do not permit the use of symlinks to tables. (This capability can be disabled with the --skip-symbolic-links [393] option.) This is especially important if you run mysqld as root, because anyone that has write access to the server's data directory then could delete any file in the system! See Section 7.10.2, “Using Symbolic Links for Tables on Unix”.
- If you do not trust your DNS, you should use IP addresses rather than host names in the grant tables. In any case, you should be very careful about creating grant table entries using host name values that contain wildcards.
- If you want to restrict the number of connections permitted to a single account, you can do so by setting the max_user_connections [420] variable in mysqld. The GRANT statement also supports resource control options for limiting the extent of server use permitted to an account. See Section 12.4.1.2, “GRANT Syntax”.
- If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using SELECT ... INTO DUMPFILE. This can be prevented by making plugin_dir [423] read only to the server.

5.4.4 Security-Related mysqld Options

The following mysqld options affect security:

Table 5.5 Security Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
allow-suspicious-udfs [384]	Yes	Yes				
chroot [385]	Yes	Yes				
des-key-file [386]	Yes	Yes				
local_infile [417]			Yes		Global	Yes
old_passwords [423]			Yes		Both	Yes
safe-show-database [391]	Yes	Yes	Yes		Global	Yes
safe-user-create [392]	Yes	Yes				
secure-auth [392]	Yes	Yes			Global	Yes
- *Variable*: secure_auth [426]			Yes		Global	Yes
skip-grant-tables [392]	Yes	Yes				
skip-name-resolve [393]	Yes	Yes			Global	No
- *Variable*: skip_name_resolve			Yes		Global	No
skip-networking [393]	Yes	Yes			Global	No
- *Variable*: skip_networking [427]			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
skip-show-database [394]	Yes	Yes			Global	No
- *Variable*: skip_show_database [427]			Yes		Global	No

- `--allow-suspicious-udfs` [384]

  This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is turned off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. This option was added in MySQL 4.0.24 and 4.1.10a. See Section 18.2.2.6, “User-Defined Function Security Precautions”.

- `--local-infile[={0|1}]` [486]

  If you start the server with `--local-infile=0` [486], clients cannot use `LOCAL` in `LOAD DATA` statements. See Section 5.4.5, “Security Issues with `LOAD DATA LOCAL`”.

- `--old-passwords` [391]

  Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See Section 5.4.2.3, “Password Hashing in MySQL”.

- `--safe-user-create` [392]

  If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` [492] privilege for the `mysql.user` table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

  ```
  GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
  ```

  This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth` [392]

  Disallow authentication for accounts that have old (pre-4.1) passwords. This option is available as of MySQL 4.1.1.

  The `mysql` client also has a `--secure-auth` [392] option, which prevents connections to a server if the server requires a password in old format for the client account.

- `--skip-grant-tables` [392]

  This option causes the server not to use the privilege system at all. This gives anyone with access to the server *unrestricted access* to *all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement. This option also suppresses loading of user-defined functions (UDFs).

- `--skip-name-resolve` [393]

Host names are not resolved. All Host column values in the grant tables must be IP addresses or localhost.

- --skip-networking [393]

  Do not permit TCP/IP connections over the network. All connections to mysqld must be made using Unix socket files. This option is unsuitable when using a MySQL version prior to 3.23.27 with the MIT-pthreads package, because Unix socket files were not supported by MIT-pthreads at that time.

- --skip-show-database [394]

  With this option, the SHOW DATABASES statement is permitted only to users who have the SHOW DATABASES [492] privilege, and the statement displays all database names. Without this option, SHOW DATABASES is permitted to all users, but displays each database name only if the user has the SHOW DATABASES [492] privilege or some privilege for the database. Note that any global privilege is a privilege for the database.

- --ssl*

  Options that begin with --ssl [521] specify whether to permit clients to connect using SSL and indicate where to find SSL keys and certificates. See Section 5.6.6.3, “SSL Command Options”.

5.4.5 Security Issues with LOAD DATA LOCAL

The LOAD DATA statement can load a file that is located on the server host, or it can load a file that is located on the client host when the LOCAL keyword is specified.

There are two potential security issues with supporting the LOCAL version of LOAD DATA statements:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the LOAD DATA statement. Such a server could access any file on the client host to which the client user has read access.
- In a Web environment where the clients are connecting from a Web server, a user could use LOAD DATA LOCAL to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the remote program being run by the user who connects to the Web server.

To deal with these problems, we changed how LOAD DATA LOCAL is handled as of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows):

- By default, all MySQL clients and libraries in binary distributions are compiled with the --enable-local-infile option, to be compatible with MySQL 3.23.48 and before.
- If you build MySQL from source but do not invoke configure with the --enable-local-infile option, LOAD DATA LOCAL cannot be used by any client unless it is written explicitly to invoke mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0). See Section 17.6.6.47, “mysql_options()”.
- You can disable all LOAD DATA LOCAL statements from the server side by starting mysqld with the --local-infile=0 [486] option.
- For the mysql command-line client, enable LOAD DATA LOCAL by specifying the --local-infile[=1] [261] option, or disable it with the --local-infile=0 [261] option. For mysqlimport,

local data file loading is off by default; enable it with the `--local` [304] or `-L` option. In any case, successful use of a local load operation requires that the server permits it.

- If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add the `local-infile=1` option to that group. However, to keep this from causing problems for programs that do not understand `local-infile`, specify it using the `loose-` prefix:

```
[client]
loose-local-infile=1
```

  The `loose-` prefix can be used as of MySQL 4.0.2.

- If `LOAD DATA LOCAL` is disabled, either in the server or the client, a client that attempts to issue such a statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

5.4.6 How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account beginning with MySQL 4.0.17 and 4.1.2. (Older MySQL versions required you to have administrator rights. This was a bug introduced in MySQL 3.23.54.)

On Unix, the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user *`user_name`*, you must do the following:

1. Stop the server if it is running (use `mysqladmin shutdown`).
2. Change the database directories and files so that *`user_name`* has privileges to read and write files in them (you might need to do this as the Unix `root` user):

```
shell> chown -R user_name /path/to/mysql/datadir
```

   If you do not do this, the server will not be able to access databases or tables when it runs as *`user_name`*.

   If directories or files within the MySQL data directory are symbolic links, `chown -R` might not follow symbolic links for you. If it does not, you will also need to follow those links and change the directories and files they point to.
3. Start the server as user *`user_name`*. Another alternative is to start `mysqld` as the Unix `root` user and use the `--user=user_name` [395] option. `mysqld` starts up, then switches to run as the Unix user *`user_name`* before accepting any connections.
4. To start the server as the given user automatically at system startup time, specify the user name by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

```
[mysqld]
user=user_name
```

If your Unix machine itself is not secured, you should assign passwords to the MySQL `root` accounts in the grant tables. Otherwise, any user with a login account on that machine can run the `mysql` client with a `--user=root` [265] option and perform any operation. (It is a good idea to assign passwords to

MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See Section 2.10, “Postinstallation Setup and Testing”.

5.5 The MySQL Access Privilege System

The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Additional functionality includes the ability to have anonymous users and to grant privileges for MySQL-specific functions such as `LOAD DATA INFILE` and administrative operations.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database or table.

The user interface to the MySQL privilege system consists of SQL statements such as `GRANT` and `REVOKE`. See Section 12.4.1, “Account Management Statements”.

Internally, the server stores privilege information in the grant tables of the `mysql` database (that is, in the database named `mysql`). The MySQL server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables.

The MySQL privilege system ensures that all users may perform only the operations permitted to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the user name you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your host name and user name in identifying you because there is no reason to assume that a given user name belongs to the same person on all hosts. For example, the user `joe` who connects from `office.example.com` need not be the same person as the user `joe` who connects from `home.example.com`. MySQL handles this by enabling you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.example.com`, and a different set of privileges for connections by `joe` from `home.example.com`. To see what privileges a given account has, use the `SHOW GRANTS` statement. For example:

```
SHOW GRANTS FOR 'joe'@'office.example.com';
SHOW GRANTS FOR 'joe'@'home.example.com';
```

MySQL access control involves two stages when you run a client program that connects to the server:

Stage 1: The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.

Stage 2: Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the `SELECT` [492] privilege for the table or the `DROP` [491] privilege for the database.

For a more detailed description of what happens during each stage, see Section 5.5.4, “Access Control, Stage 1: Connection Verification”, and Section 5.5.5, “Access Control, Stage 2: Request Verification”.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. For details about the conditions under which the server reloads the grant tables, see Section 5.5.6, “When Privilege Changes Take Effect”.

For general security-related advice, see Section 5.4, “General Security Issues”. For help in diagnosing privilege-related problems, see Section 5.5.7, “Causes of Access-Denied Errors”.

5.5.1 Privileges Provided by MySQL

MySQL provides privileges that apply in different contexts and at different levels of operation:

- Administrative privileges enable users to manage operation of the MySQL server. These privileges are global because they are not specific to a particular database.
- Database privileges apply to a database and to all objects within it. These privileges can be granted for specific databases, or globally so that they apply to all databases.
- Privileges for database objects such as tables, indexes, views, and stored routines can be granted for specific objects within a database, for all objects of a given type within a database (for example, all tables in a database), or globally for all objects of a given type in all databases).

Information about account privileges is stored in the `user`, `db`, `host`, `tables_priv`, and `columns_priv` tables in the `mysql` database (see Section 5.5.2, “Privilege System Grant Tables”). The MySQL server reads the contents of these tables into memory when it starts and reloads them under the circumstances indicated in Section 5.5.6, “When Privilege Changes Take Effect”. Access-control decisions are based on the in-memory copies of the grant tables.

Some releases of MySQL introduce changes to the structure of the grant tables to add new access privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

The following table shows the privilege names used at the SQL level in the `GRANT` and `REVOKE` statements, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies.

Table 5.6 Permissible Privileges for `GRANT` and `REVOKE`

Privilege	Column	Context
`CREATE` [491]	`Create_priv`	databases, tables, or indexes
`DROP` [491]	`Drop_priv`	databases or tables
`GRANT OPTION` [491]	`Grant_priv`	databases, tables, or stored routines
`REFERENCES` [492]	`References_priv`	databases or tables
`ALTER` [491]	`Alter_priv`	tables
`DELETE` [491]	`Delete_priv`	tables
`INDEX` [491]	`Index_priv`	tables
`INSERT` [492]	`Insert_priv`	tables or columns
`SELECT` [492]	`Select_priv`	tables or columns
`UPDATE` [493]	`Update_priv`	tables or columns
`CREATE TEMPORARY TABLES` [491]	`Create_tmp_table_priv`	tables

Privilege	Column	Context
LOCK TABLES [492]	Lock_tables_priv	tables
FILE [491]	File_priv	file access on server host
PROCESS [492]	Process_priv	server administration
RELOAD [492]	Reload_priv	server administration
REPLICATION CLIENT [492]	Repl_client_priv	server administration
REPLICATION SLAVE [492]	Repl_slave_priv	server administration
SHOW DATABASES [492]	Show_db_priv	server administration
SHUTDOWN [492]	Shutdown_priv	server administration
SUPER [493]	Super_priv	server administration
ALL [PRIVILEGES] [491]		server administration
USAGE [493]		server administration

The following list provides a general description of each privilege available in MySQL. Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- The ALL [491] or ALL PRIVILEGES [491] privilege specifier is shorthand. It stands for “all privileges available at a given privilege level” (except GRANT OPTION [491]). For example, granting ALL [491] at the global or table level grants all global privileges or all table-level privileges.
- The ALTER [491] privilege enables use of ALTER TABLE to change the structure of or rename tables. (ALTER TABLE also requires the INSERT [492] and CREATE [491] privileges.)
- The CREATE [491] privilege enables creation of new databases and tables.
- The CREATE TEMPORARY TABLES [491] privilege enables the use of the keyword TEMPORARY in CREATE TABLE statements. This privilege was added in MySQL 4.0.2.
- The DELETE [491] privilege enables rows to be deleted from tables in a database.
- The DROP [491] privilege enables you to drop (remove) existing databases and tables. If you grant the DROP [491] privilege for the mysql database to a user, that user can drop the database in which the MySQL access privileges are stored!
- The EXECUTE [491] privilege was added in MySQL 4.0.2, but is not used until MySQL 5.0.
- The FILE [491] privilege gives you permission to read and write files on the server host using the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements and the LOAD_FILE() [798] function. A user who has the FILE [491] privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The FILE [491] privilege also enables the user to create new files in any directory where the MySQL server has write access. This includes the server's data directory containing the files that implement the privilege tables. As a security measure, the server will not overwrite existing files.
- The GRANT OPTION [491] privilege enables you to give to other users or remove from other users those privileges that you yourself possess.
- The INDEX [491] privilege enables you to create or drop (remove) indexes. INDEX [491] applies to existing tables. If you have the CREATE [491] privilege for a table, you can include index definitions in the CREATE TABLE statement.

- The `INSERT` [492] privilege enables rows to be inserted into tables in a database. `INSERT` [492] is also required for the `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` table-maintenance statements.

- The `LOCK TABLES` [492] privilege enables the use of explicit `LOCK TABLES` statements to lock tables for which you have the `SELECT` [492] privilege. This includes the use of write locks, which prevents other sessions from reading the locked table. This privilege was added in MySQL 4.0.2.

- The `PROCESS` [492] privilege pertains to display of information about the threads executing within the server (that is, information about the statements being executed by sessions). The privilege enables use of `SHOW PROCESSLIST` or `mysqladmin processlist` to see threads belonging to other accounts; you can always see your own threads. Prior to MySQL 4.0.2, `PROCESS` [492] also enable the use of `KILL` to kill threads belonging to other accounts; you can always kill your own threads.

- The `REFERENCES` [492] privilege currently is unused.

- The `RELOAD` [492] privilege enables use of the `FLUSH` statement. It also enables `mysqladmin` commands that are equivalent to `FLUSH` operations: `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables`, `flush-threads`, `refresh`, and `reload`.

  The `reload` command tells the server to reload the grant tables into memory. `flush-privileges` is a synonym for `reload`. The `refresh` command closes and reopens the log files and flushes all tables. The other `flush-`*`xxx`* commands perform functions similar to `refresh`, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, `flush-logs` is a better choice than `refresh`.

- The `REPLICATION CLIENT` [492] privilege enables the use of `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`. This privilege was added in MySQL 4.0.2.

- The `REPLICATION SLAVE` [492] privilege should be granted to accounts that are used by slave servers to connect to the current server as their master. Without this privilege, the slave cannot request updates that have been made to databases on the master server. This privilege was added in MySQL 4.0.2.

- The `SELECT` [492] privilege enables you to select rows from tables in a database. `SELECT` statements require the `SELECT` [492] privilege only if they actually retrieve rows from a table. Some `SELECT` statements do not access tables and can be executed without permission for any database. For example, you can use `SELECT` as a simple calculator to evaluate expressions that make no reference to tables:

  ```
  SELECT 1+1;
  SELECT PI()*2;
  ```

  The `SELECT` [492] privilege is also needed for other statements that read column values. For example, `SELECT` [492] is needed for columns referenced on the right hand side of *`col_name`*=*`expr`* assignment in `UPDATE` statements or for columns named in the `WHERE` clause of `DELETE` or `UPDATE` statements.

- The `SHOW DATABASES` [492] privilege enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` [394] option. Note that *any* global privilege is a privilege for the database. `SHOW DATABASES` [492] was added in MySQL 4.0.2.

- The `SHUTDOWN` [492] privilege enables use of the `mysqladmin shutdown` command. There is no corresponding SQL statement.

- The SUPER [493] privilege enables an account to use CHANGE MASTER TO, KILL or mysqladmin kill to kill threads belonging to other accounts (you can always kill your own threads), PURGE BINARY LOGS, configuration changes using SET GLOBAL to modify global system variables, the mysqladmin debug command, enabling or disabling logging, performing updates even if the read_only [425] system variable is enabled, starting and stopping replication on slave servers, and enables you to connect (once) even if the connection limit controlled by the max_connections [419] system variable is reached. This privilege was added in MySQL 4.0.2. Prior to MySQL 4.0.2, the PROCESS [492] privilege controls the ability to terminate threads for other accounts.
- The UPDATE [493] privilege enables rows to be updated in tables in a database.
- The USAGE [493] privilege specifier stands for “no privileges.” It is used at the global level with GRANT to modify account attributes such as resource limits or SSL characteristics without affecting existing account privileges.

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the FILE [491] and administrative privileges:

- The FILE [491] privilege can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using SELECT to transfer its contents to the client host.
- The GRANT OPTION [491] privilege enables users to give their privileges to other users. Two users that have different privileges and with the GRANT OPTION [491] privilege are able to combine privileges.
- The ALTER [491] privilege may be used to subvert the privilege system by renaming tables.
- The SHUTDOWN [492] privilege can be abused to deny service to other users entirely by terminating the server.
- The PROCESS [492] privilege can be used to view the plain text of currently executing statements, including statements that set or change passwords.
- The SUPER [493] privilege can be used to terminate other sessions or change how the server operates.
- Privileges granted for the mysql database itself can be used to change passwords and other access privilege information. Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the user table Password column can change an account's password, and then connect to the MySQL server using that account.

5.5.2 Privilege System Grant Tables

Normally, you manipulate the contents of the grant tables in the mysql database indirectly by using statements such as GRANT and REVOKE to set up accounts and control the privileges available to each one. See Section 12.4.1, “Account Management Statements”. The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients.

These mysql database tables contain grant information:

- user: Contains user accounts, global privileges, and other non-privilege columns.
- db: Contains database-level privileges.
- host: Obsolete.

- `tables_priv`: Contains table-level privileges.
- `columns_priv`: Contains column-level privileges.

Other tables in the `mysql` database do not hold grant information and are discussed elsewhere:

- `func`: Contains information about user-defined functions: See Section 18.2, “Adding New Functions to MySQL”.
- `help_xxx`: These tables are used for server-side help: See Section 5.1.7, “Server-Side Help”.
- `time_zone_xxx`: These tables contain time zone information: See Section 9.7, “MySQL Server Time Zone Support”.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row (entry) in the tables; that is, the context in which the row applies. For example, a `user` table row with `Host` and `User` values of `'thomas.loc.gov'` and `'bob'` would be used for authenticating connections made to the server from the host `thomas.loc.gov` by a client that specifies a user name of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'thomas.loc.gov'`, `'bob'` and `'reports'` would be used when `bob` connects from the host `thomas.loc.gov` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies.
- Privilege columns indicate which privileges are granted by a table row; that is, what operations can be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. Section 5.5.5, “Access Control, Stage 2: Request Verification”, describes the rules that are used to do this.

The server uses the grant tables in the following manner:

- The `user` table scope columns determine whether to reject or permit incoming connections. For permitted connections, any privileges granted in the `user` table indicate the user's global privileges. Any privilege granted in this table applies to *all* databases on the server.

Note

Because any global privilege is considered a privilege for all databases, any global privilege enables a user to see all database names with `SHOW DATABASES`.

- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine which operations are permitted. A privilege granted at the database level applies to the database and to all objects in the database, such as tables and stored programs.
- The `host` table is used in conjunction with the `db` table when you want a given `db` table row to apply to several hosts. For example, if you want a user to be able to use a database from several hosts in your network, leave the `Host` value empty in the user's `db` table row, then populate the `host` table with a row for each of those hosts. This mechanism is described more detail in Section 5.5.5, “Access Control, Stage 2: Request Verification”.

Note

The `host` table must be modified directly with statements such as `INSERT`, `UPDATE`, and `DELETE`. It is not affected by statements such as `GRANT` and `REVOKE` that modify the grant tables indirectly. Most MySQL installations need not use this table at all.

- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.

The server uses the `user`, `db`, and `host` tables in the `mysql` database at both the first and second stages of access control (see Section 5.5, "The MySQL Access Privilege System"). The columns in the `user` and `db` tables are shown here. The `host` table is similar to the `db` table but has a specialized use as described in Section 5.5.5, "Access Control, Stage 2: Request Verification".

Table 5.7 `user` and `db` Table Columns

Table Name	`user`	`db`
Scope columns	`Host`	`Host`
	`User`	`Db`
	`Password`	`User`
Privilege columns	`Select_priv`	`Select_priv`
	`Insert_priv`	`Insert_priv`
	`Update_priv`	`Update_priv`
	`Delete_priv`	`Delete_priv`
	`Index_priv`	`Index_priv`
	`Alter_priv`	`Alter_priv`
	`Create_priv`	`Create_priv`
	`Drop_priv`	`Drop_priv`
	`Grant_priv`	`Grant_priv`
	`References_priv`	`References_priv`
	`Execute_priv`	
	`Reload_priv`	
	`Shutdown_priv`	
	`Process_priv`	
	`File_priv`	
	`Show_db_priv`	
	`Super_priv`	
	`Create_tmp_table_priv`	`Create_tmp_table_priv`
	`Lock_tables_priv`	`Lock_tables_priv`
	`Repl_slave_priv`	
	`Repl_client_priv`	
Security columns	`ssl_type`	
	`ssl_cipher`	
	`x509_issuer`	
	`x509_subject`	
Resource control columns	`max_questions`	
	`max_updates`	

Table Name	user	db
	max_connections	
	max_user_connections	

The ssl_type, ssl_cipher, x509_issuer, and x509_subject columns were added in MySQL 4.0.0.

The Create_tmp_table_priv, Execute_priv, Lock_tables_priv, Repl_client_priv, Repl_slave_priv, Show_db_priv, Super_priv, max_questions, max_updates, and max_connections columns were added in MySQL 4.0.2. Execute_priv is not operational through MySQL 4.1.

During the second stage of access control, the server performs request verification to make sure that each client has sufficient privileges for each request that it issues. In addition to the user, db, and host grant tables, the server may also consult the tables_priv and columns_priv tables for requests that involve tables. The latter tables provide finer privilege control at the table and column levels. They have the columns shown in the following table.

Table 5.8 tables_priv and columns_priv Table Columns

Table Name	tables_priv	columns_priv
Scope columns	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Privilege columns	Table_priv	Column_priv
	Column_priv	
Other columns	Timestamp	Timestamp
	Grantor	

The Timestamp and Grantor columns are set to the current timestamp and the CURRENT_USER [872] value, respectively. However, they are unused and are discussed no further here.

Scope columns in the grant tables contain strings. They are declared as shown here; the default value for each is the empty string.

Table 5.9 Grant Table Scope Column Types

Column Name	Type
Column Name	**Type**
Host	CHAR(60)
User	CHAR(16)
Password	CHAR(41)
Db	CHAR(64)
Table_name	CHAR(64)
Column_name	CHAR(64)
Routine_name	CHAR(64)

Before MySQL 3.23, the `Db` column is `CHAR(32)` in some tables and `CHAR(60)` in others.

For access-checking purposes, comparisons of `User`, `Password`, `Db`, and `Table_name` values are case sensitive. Comparisons of `Host` values are not case sensitive. Comparisons of `Column_name` values are not case sensitive as of MySQL 3.22.12.

In the `user`, `db`, and `host` tables, each privilege is listed in a separate column that is declared as `ENUM('N','Y') DEFAULT 'N'`. In other words, each privilege can be disabled or enabled, with the default being disabled.

In the `tables_priv` and `columns_priv` tables, the privilege columns are declared as `SET` columns. Values in these columns can contain any combination of the privileges controlled by the table. Only those privileges listed in the column value are enabled.

Table 5.10 Set-Type Privilege Column Values

Table Name	Column Name	Possible Set Elements
Table Name	**Column Name**	**Possible Set Elements**
`tables_priv`	`Table_priv`	`'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'`
`tables_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`columns_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`

Administrative privileges (such as `RELOAD` [492] or `SHUTDOWN` [492]) are specified only in the `user` table. Administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. Consequently, to determine whether you can perform an administrative operation, the server need consult only the `user` table.

The `FILE` [491] privilege also is specified only in the `user` table. It is not an administrative privilege as such, but your ability to read or write files on the server host is independent of the database you are accessing.

The `mysqld` server reads the contents of the grant tables into memory when it starts. You can tell it to reload the tables by issuing a `FLUSH PRIVILEGES` statement or executing a `mysqladmin flush-privileges` or `mysqladmin reload` command. Changes to the grant tables take effect as indicated in Section 5.5.6, “When Privilege Changes Take Effect”.

When you modify an account's privileges, it is a good idea to verify that the changes set up privileges the way you want. To check the privileges for a given account, use the `SHOW GRANTS` statement (see Section 12.4.5.12, “`SHOW GRANTS` Syntax”). For example, to determine the privileges that are granted to an account with user name and host name values of `bob` and `pc84.example.com`, use this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

5.5.3 Specifying Account Names

MySQL account names consist of a user name and a host name. This enables creation of accounts for users with the same name who can connect from different hosts. This section describes how to write account names, including special values and wildcard rules.

In SQL statements such as `GRANT` and `SET PASSWORD`, write account names using the following rules:

- Syntax for account names is *'user_name'@'host_name'*.
- An account name consisting only of a user name is equivalent to *'user_name'@'%'*. For example, 'me' is equivalent to 'me'@'%'.
- The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes are necessary to specify a *user_name* string containing special characters (such as "-"), or a *host_name* string containing special characters or wildcard characters (such as "%"); for example, 'test-user'@'%.com'.
- Quote user names and host names as identifiers or as strings, using either backticks ("`"), single quotation marks ("'"), or double quotation marks ("").
- The user name and host name parts, if quoted, must be quoted separately. That is, write 'me'@'localhost', not 'me@localhost'; the latter is interpreted as 'me@localhost'@'%'.

MySQL stores account names in grant tables in the mysql database using separate columns for the user name and host name parts:

- The user table contains one row for each account. The User and Host columns store the user name and host name. This table also indicates which global privileges the account has.
- Other grant tables indicate privileges an account has for databases and objects within databases. These tables have User and Host columns to store the account name. Each row in these tables associates with the account in the user table that has the same User and Host values.

For additional detail about grant table structure, see Section 5.5.2, "Privilege System Grant Tables".

User names and host names have certain special values or wildcard conventions, as described following.

A user name is either a nonblank value that literally matches the user name for incoming connection attempts, or a blank value (empty string) that matches any user name. An account with a blank user name is an anonymous user. To specify an anonymous user in SQL statements, use a quoted empty user name part, such as ''@'localhost'.

The host name part of an account name can take many forms, and wildcards are permitted:

- A host value can be a host name or an IP address. The name 'localhost' indicates the local host. The IP address '127.0.0.1' indicates the loopback interface.
- You can use the wildcard characters "%" and "_" in host values. These have the same meaning as for pattern-matching operations performed with the LIKE [804] operator. For example, a host value of '%' matches any host name, whereas a value of '%.mysql.com' matches any host in the mysql.com domain. '192.168.1.%' matches any host in the 192.168.1 class C network.

  Because you can use IP wildcard values in host values (for example, '192.168.1.%' to match every host on a subnet), someone could try to exploit this capability by naming a host 192.168.1.somewhere.com. To foil such attempts, MySQL disallows matching on host names that start with digits and a dot. Thus, if you have a host named something like 1.2.example.com, its name never matches the host part of account names. An IP wildcard value can match only IP addresses, not host names.
- For a host value specified as an IP address, you can specify a netmask indicating how many address bits to use for the network number. The syntax is *host_ip/netmask*. For example:

```
GRANT ALL PRIVILEGES ON db.* TO 'david'@'192.58.197.0/255.255.255.0';
```

This enables `david` to connect from any client host having an IP address *`client_ip`* for which the following condition is true:

```
client_ip & netmask = host_ip
```

That is, for the `GRANT` statement just shown:

```
client_ip & 255.255.255.0 = 192.58.197.0
```

IP addresses that satisfy this condition and can connect to the MySQL server are those in the range from `192.58.197.0` to `192.58.197.255`.

The netmask can only be used to tell the server to use 8, 16, 24, or 32 bits of the address. Examples:

- `192.0.0.0/255.0.0.0`: Any host on the 192 class A network
- `192.168.0.0/255.255.0.0`: Any host on the 192.168 class B network
- `192.168.1.0/255.255.255.0`: Any host on the 192.168.1 class C network
- `192.168.1.1`: Only the host with this specific IP address

The following netmask will not work because it masks 28 bits, and 28 is not a multiple of 8:

```
192.168.0.1/255.255.255.240
```

5.5.4 Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL user name
- A reference to the `CURRENT_USER()` [872] (or `CURRENT_USER` [872]) function is equivalent to specifying the current user's name and host name literally.

Identity checking is performed using the three `user` table scope columns (`Host`, `User`, and `Password`). The server accepts the connection only if the `Host` and `User` columns in some `user` table row match the client host name and user name and the client supplies the password specified in that row. The rules for permissible `Host` and `User` values are given in Section 5.5.3, “Specifying Account Names”.

If the `User` column value is nonblank, the user name in an incoming connection must match exactly. If the `User` value is blank, it matches any user name. If the `user` table row that matches an incoming connection has a blank user name, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during Stage 2).

The `Password` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password.

Nonblank `Password` values in the `user` table represent encrypted passwords. MySQL does not store passwords in plaintext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the `PASSWORD()` [868] function). The encrypted password then is used during the connection process when checking whether the password is correct. (This is done without the encrypted password ever traveling over the connection.) See Section 5.6.1, “User Names and Passwords”.

From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give nonadministrative users read access to tables in the `mysql` database*.

The following table shows how various combinations of `Host` and `User` values in the `user` table apply to incoming connections.

`Host` Value	**`User` Value**	**Permissible Connections**
`'thomas.loc.gov'`	`'fred'`	`fred`, connecting from `thomas.loc.gov`
`'thomas.loc.gov'`	`''`	Any user, connecting from `thomas.loc.gov`
`'%'`	`'fred'`	`fred`, connecting from any host
`'%'`	`''`	Any user, connecting from any host
`'%.loc.gov'`	`'fred'`	`fred`, connecting from any host in the `loc.gov` domain
`'x.y.%'`	`'fred'`	`fred`, connecting from `x.y.net`, `x.y.com`, `x.y.edu`, and so on; this is probably not useful
`'144.155.166.177'`	`'fred'`	`fred`, connecting from the host with IP address `144.155.166.177`
`'144.155.166.%'`	`'fred'`	`fred`, connecting from any host in the `144.155.166` class C subnet
`'144.155.166.0/255.255.255.0"fred'`		Same as previous example

It is possible for the client host name and user name of an incoming connection to match more than one row in the `user` table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from `thomas.loc.gov` by `fred`.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the `user` table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client host name and user name.

To see how this works, suppose that the `user` table looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

When the server reads the table into memory, it orders the rows with the most-specific `Host` values first. Literal host names and IP addresses are the most specific. (The specificity of a literal IP address is not affected by whether it has a netmask, so `192.168.1.13` and `192.168.1.0/255.255.255.0` are considered equally specific.) The pattern `'%'` means “any host” and is least specific. Rows with the same `Host` value are ordered with the most-specific `User` values first (a blank `User` value means “any user” and is least specific). For the `user` table just shown, the result after sorting looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with `Host` and `User` values of `'localhost'` and `''`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| thomas.loc.gov |          | ...
+----------------+----------+-
```

The sorted table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| thomas.loc.gov |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

A connection by `jeffrey` from `thomas.loc.gov` is matched by the first row, whereas a connection by `jeffrey` from any host is matched by the second.

Note

It is a common misconception to think that, for a given user name, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is not true. The preceding example illustrates this, where a connection from `thomas.loc.gov` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no user name. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a user name when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` [872] function. (See Section 11.13, “Information Functions”.) It returns a value in *`user_name@host_name`* format that indicates the `User` and `Host` values from the matching `user` table row. Suppose that `jeffrey` connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

The result shown here indicates that the matching `user` table row had a blank `User` column value. In other words, the server is treating `jeffrey` as an anonymous user.

The `CURRENT_USER()` [872] function is available as of MySQL 4.0.6. See Section 11.13, "Information Functions". Another way to diagnose authentication problems is to print out the `user` table and sort it by hand to see where the first match is being made.

5.5.5 Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue through that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `db`, `host`, `tables_priv`, or `columns_priv` tables. (You may find it helpful to refer to Section 5.5.2, "Privilege System Grant Tables", which lists the columns present in each of the grant tables.)

The `user` table grants privileges that are assigned to you on a global basis and that apply no matter what the default database is. For example, if the `user` table grants you the `DELETE` [491] privilege, you can delete rows from any table in any database on the server host! It is wise to grant privileges in the `user` table only to people who need them, such as database administrators. For other users, you should leave all privileges in the `user` table set to `'N'` and grant privileges at more specific levels only. You can grant privileges for particular databases, tables, or columns.

The `db` and `host` tables grant database-specific privileges. Values in the scope columns of these tables can take the following forms:

- A blank `User` value in the `db` table matches the anonymous user. A nonblank value matches literally; there are no wildcards in user names.
- The wildcard characters "`%`" and "`_`" can be used in the `Host` and `Db` columns of either table. These have the same meaning as for pattern-matching operations performed with the `LIKE` [804] operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include "`_`" character as part of a database name, specify it as "`\_`" in the `GRANT` statement.
- A `'%'` `Host` value in the `db` table means "any host." A blank `Host` value in the `db` table means "consult the `host` table for further information" (a process that is described later in this section).
- A `'%'` or blank `Host` value in the `host` table means "any host."
- A `'%'` or blank `Db` value in either table means "any database."

The server reads the `db` and `host` tables into memory and sorts them at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns, and sorts the `host` table based on the `Host` and `Db` scope columns. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching entries, it uses the first match that it finds.

The `tables_priv` and `columns_priv` tables grant table-specific and column-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters “`%`” and “`_`” can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE` [804] operator.
- A `'%'` or blank `Host` value means “any host.”
- The `Db`, `Table_name`, and `Column_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv` and `columns_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN` [492] or `RELOAD` [492], the server checks only the `user` table row because that is the only table that specifies administrative privileges. The server grants access if the row permits the requested operation and denies access otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table row does not grant the `SHUTDOWN` [492] privilege to you, the server denies access without even checking the `db` or `host` tables. (They contain no `Shutdown_priv` column, so there is no need to do so.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global privileges by looking in the `user` table row. If the row permits the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges by checking the `db` and `host` tables:

1. The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user's host name and MySQL user name. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.
2. If there is a matching `db` table row and its `Host` column is not blank, that row defines the user's database-specific privileges.
3. If the matching `db` table row's `Host` column is blank, it signifies that the `host` table enumerates which hosts should be permitted access to the database. In this case, a further lookup is done in the `host` table to find a match on the `Host` and `Db` columns. If no `host` table row matches, access is denied. If there is a match, the user's database-specific privileges are computed as the intersection (*not* the union!) of the privileges in the `db` and `host` table entries; that is, the privileges that are `'Y'` in both entries. (This way you can grant general privileges in the `db` table row and then selectively restrict them on a host-by-host basis using the `host` table entries.)

After determining the database-specific privileges granted by the `db` and `host` table entries, the server adds them to the global privileges granted by the `user` table. If the result permits the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user's privileges, and permits or denies access based on the result.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

It may not be apparent why, if the global `user` row privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute

an `INSERT INTO ... SELECT` statement, you need both the `INSERT` [492] and the `SELECT` [492] privileges. Your privileges might be such that the `user` table row grants one privilege and the `db` table row grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by the entries in both tables must be combined.

The `host` table is not affected by the `GRANT` or `REVOKE` statements, so it is unused in most MySQL installations. If you modify it directly, you can use it for some specialized purposes, such as to maintain a list of secure servers on the local network that are granted all privileges.

You can also use the `host` table to indicate hosts that are *not* secure. Suppose that you have a machine `public.your.domain` that is located in a public area that you do not consider secure. You can enable access to all hosts on your network except that machine by using `host` table entries like this:

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (all privileges set to 'N')
| %.your.domain      | %  | ... (all privileges set to 'Y')
+--------------------+----+-
```

5.5.6 When Privilege Changes Take Effect

When `mysqld` starts, it reads all grant table contents into memory. The in-memory tables become effective for access control at that point.

If you modify the grant tables indirectly using account-management statements such as `GRANT`, `REVOKE`, or `SET PASSWORD`, the server notices these changes and loads the grant tables into memory again immediately.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE`, your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables. If you change the grant tables directly but forget to reload them, your changes have *no effect* until you restart the server. This may leave you wondering why your changes seem to make no difference!

To tell the server to reload the grant tables, perform a flush-privileges operation. This can be done by issuing a `FLUSH PRIVILEGES` statement or by executing a `mysqladmin flush-privileges` or `mysqladmin reload` command.

A grant table reload affects privileges for each existing client connection as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect the next time the client executes a `USE db_name` statement.

Note

Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database or flushing the privileges.

- Global privileges and passwords are unaffected for a connected client. These changes take effect only for subsequent connections.

If the server is started with the `--skip-grant-tables` [392] option, it does not read the grant tables or implement any access control. Anyone can connect and do anything, *which is insecure.* To cause a server thus started to read the tables and enable access checking, flush the privileges.

5.5.7 Causes of Access-Denied Errors

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not, clients cannot connect to it. For example, if an attempt to connect to the server fails with a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

- It might be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` [226] option to indicate the proper port number, or a `--socket` [227] option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking` [393], it will not accept TCP/IP connections at all. If the server was started with `--bind-address=127.0.0.1` [384], it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux), the installation process initializes the `mysql` database containing the grant tables. For distributions that do not do this, you must initialize the grant tables manually by running the `mysql_install_db` script. For details, see Section 2.10.2, “Unix Postinstallation Procedures”.

  To determine whether you need to initialize the grant tables, look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If not, execute the `mysql_install_db` script. After running this script and starting the server, test the initial privileges by executing this command:

```
shell> mysql -u root test
```

  The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

```
shell> mysql -u root mysql
```

The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL accounts. For instructions on setting the initial passwords, see Section 2.10.3, “Securing the Initial MySQL Accounts”.

- If you have updated an existing MySQL installation to a newer version, did you run the `mysql_fix_privilege_tables` script? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

  For information on how to deal with this, see Section 5.4.2.3, “Password Hashing in MySQL”, and Section B.5.2.4, “`Client does not support authentication protocol`”.

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check any applicable option files and your environment. For example, if you get `Access denied` when you run a client without any options, make sure that you have not specified an old password in any of your option files!

  You can suppress the use of option files by a client program by invoking it with the `--no-defaults` [235] option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

  The option files that clients use are listed in Section 4.2.3.3, “Using Option Files”. Environment variables are listed in Section 2.13, “Environment Variables”.

- If you get the following error, it means that you are using an incorrect `root` password:

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

  If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in some option file. Try the `--no-defaults` [235] option as described in the previous item.

  For information on changing passwords, see Section 5.6.5, “Assigning Account Passwords”.

  If you have lost or forgotten the `root` password, see Section B.5.4.1, “How to Reset the Root Password”.

- If you change a password by using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must encrypt the password using the `PASSWORD()` [868] function. If you do not use `PASSWORD()` [868] for these statements, the password will not work. For example, the following statement assigns a password, but fails to encrypt it, so the user is not able to connect afterward:

```
```

```
SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

Instead, set the password like this:

```
SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

The `PASSWORD()` [868] function is unnecessary when you specify a password using the `GRANT` statement or the `mysqladmin password` command. Each of those automatically uses `PASSWORD()` [868] to encrypt the password. See Section 5.6.5, “Assigning Account Passwords”.

- `localhost` is a synonym for your local host name, and is also the default host to which clients try to connect if you specify no host explicitly. However, connections to `localhost` on Unix systems do not work if you are using a MySQL version older than 3.23.27 that uses MIT-pthreads: `localhost` connections are made using Unix socket files, which were not supported by MIT-pthreads at that time.

  To avoid this problem on such systems, you can use a `--host=127.0.0.1` [226] option to name the server host explicitly. This will make a TCP/IP connection to the local `mysqld` server. You can also use TCP/IP by specifying a `--host` [226] option that uses the actual host name of the local host. In this case, the host name must be specified in a `user` table row on the server host, even though you are running the client program on the same host as the server.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the `user` table that exactly matches the host name and user name that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without a password.

- If you get an `Access denied` error when trying to connect to the database with `mysql -u user_name`, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

  ```
  SELECT * FROM user;
  ```

  The result should include a row with the `Host` and `User` columns matching your client's host name and your MySQL user name.

- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the `user` table with a `Host` value that matches the client host:

  ```
  Host ... is not allowed to connect to this MySQL server
  ```

  You can fix this by setting up an account for the combination of client host name and user name that you are using when trying to connect.

  If you do not know the IP address or host name of the machine from which you are connecting, you should put a row with `'%'` as the `Host` column value in the `user` table. After trying to connect from the client machine, use a `SELECT USER()` query to see how you really did connect. Then change the `'%'` in the `user` table row to the actual host name that shows up in the log. Otherwise, your system is left insecure because it permits connections from any host for the given user name.

  (Note that if you are running a version of MySQL older than 3.23.11, the output from `USER()` [876] does not include the host name. In this case, you must restart the server with the `--log` [388] option, then obtain the host name from the log.)

On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

- If you specify a host name when trying to connect, but get an error message where the host name is not shown or is an IP address, it means that the MySQL server got an error when trying to resolve the IP address of the client host to a name:

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@'' (using password: YES)
```

If you try to connect as `root` and get the following error, it means that you do not have a row in the `user` table with a `User` column value of `'root'` and that `mysqld` cannot resolve the host name for your client:

```
Access denied for user ''@'unknown'
```

These errors indicate a DNS problem. To fix it, execute `mysqladmin flush-hosts` to reset the internal DNS host name cache. See Section 7.8.5, “How MySQL Uses DNS”.

Some permanent solutions are:

- Determine what is wrong with your DNS server and fix it.
- Specify IP addresses rather than host names in the MySQL grant tables.
- Put an entry for the client machine name in `/etc/hosts` on Unix or `\windows\hosts` on Windows.
- Start `mysqld` with the `--skip-name-resolve` [393] option.
- Start `mysqld` with the `--skip-host-cache` [392] option.
- On Unix, if you are running the server and the client on the same machine, connect to `localhost`. Unix connections to `localhost` use a Unix socket file rather than TCP/IP.
- On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the host name `.` (period). Connections to `.` use a named pipe rather than TCP/IP.

- If `mysql -u root test` works but `mysql -h your_hostname -u root test` results in `Access denied` (where *`your_hostname`* is the actual host name of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified host name, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have an entry with host `'pluto'` in the `user` table, but your DNS tells MySQL that your host name is `'pluto.example.com'`, the entry does not work. Try adding an entry to the `user` table that contains the IP address of your host as the `Host` column value. (Alternatively, you could add an entry to the `user` table with a `Host` value that contains a wildcard; for example, `'pluto.%'`. However, use of `Host` values ending with “`%`” is *insecure* and is *not* recommended!)

- If `mysql -u user_name test` works but `mysql -u user_name other_db` does not, you have not granted access to the given user for the database named *`other_db`*.

- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given user name from the remote host.

- If you cannot figure out why you get `Access denied`, remove from the `user` table all entries that have `Host` values containing wildcards (entries that contain `'%'` or `'_'` characters). A very common error is to insert a new entry with `Host='%'` and `User='some_user'`, thinking that this enables you to specify `localhost` to connect from the same machine. The reason that this does not work is that the default privileges include an entry with `Host='localhost'` and `User=''`. Because that entry has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new entry when connecting from `localhost`! The correct procedure is to insert a second entry with `Host='localhost'` and `User='some_user'`, or to delete the entry with `Host='localhost'` and `User=''`. After deleting the entry, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables. See also Section 5.5.4, “Access Control, Stage 1: Connection Verification”.

- If you are able to connect to the MySQL server, but get an `Access denied` message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA INFILE` statement, your entry in the `user` table does not have the `FILE` [491] privilege enabled.

- If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a `mysqladmin flush-privileges` command to cause the server to reload the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the `root` password with an `UPDATE` statement, you will not need to specify the new password until after you flush the privileges, because the server will not know you've changed the password yet!

- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 5.5.6, “When Privilege Changes Take Effect”.

- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -pyour_pass db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=your_pass` [226] syntax to specify the password. If you use the `-p` or `--password` [226] option with no password value, MySQL prompts you for the password.)

- For testing purposes, start the `mysqld` server with the `--skip-grant-tables` [392] option. Then you can change the MySQL grant tables and use the `mysqlaccess` script to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to reload the privileges. This enables you to begin using the new grant table contents without stopping and restarting the server.

- If you get the following error, you may have a problem with the `db` or `host` table:

  ```
  Access to database denied
  ```

  If the entry selected from the `db` table has an empty value in the `Host` column, make sure that there are one or more corresponding entries in the `host` table specifying which hosts the `db` table entry applies to. This problem occurs infrequently because the `host` table is rarely used.

- If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query` [385]). This prints host and user information about attempted connections, as well as information about each command issued. See Section 18.4.3, “The DBUG Package”.

- If you have any other problems with the MySQL grant tables and feel you must post the problem to the mailing list, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at Section 1.8, "How to Report Bugs or Problems". In some cases, you may need to restart `mysqld` with `--skip-grant-tables` [392] to run `mysqldump`.

5.6 MySQL User Account Management

This section describes how to set up accounts for clients of your MySQL server. It discusses the following topics:

- The meaning of account names and passwords as used in MySQL and how that compares to names and passwords used by your operating system
- How to set up new accounts and remove existing accounts
- How to change passwords
- Guidelines for using passwords securely
- How to use secure connections with SSL

See also Section 12.4.1, "Account Management Statements", which describes the syntax and use for all user-management SQL statements.

5.6.1 User Names and Passwords

MySQL stores accounts in the `user` table of the `mysql` database. An account is defined in terms of a user name and the client host or hosts from which the user can connect to the server. The account may also have a password. For information about account representation in the `user` table, see Section 5.5.2, "Privilege System Grant Tables".

There are several distinctions between the way user names and passwords are used by MySQL and the way they are used by your operating system:

- User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix user name as the MySQL user name, but that is for convenience only. The default can be overridden easily, because client programs permit any user name to be specified with a `-u` or `--user` option. Because this means that anyone can attempt to connect to the server using any user name, you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a user name for an account that has no password is able to connect successfully to the server.
- MySQL user names can be up to 16 characters long. Operating system user names, because they are completely unrelated to MySQL user names, may be of a different maximum length. For example, Unix user names typically are limited to eight characters.

Warning

The limit on MySQL user name length is hard-coded in the MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure prescribed that is described in Section 4.4.5, "`mysql_fix_privilege_tables` — Upgrade MySQL System

> Tables”. Attempting to redefine MySQL's system tables in any other fashion results in undefined (and unsupported!) behavior.

- MySQL user names can be up to 16 characters long. *Changing the maximum length is not supported*. If you try to change it, for example by changing the length of the `User` column in the `mysql` database tables, this will result in unpredictable behavior. (Altering privilege tables is not supported in any case.) Operating system user names might have a different maximum length. For example, Unix user names typically are limited to eight characters.
- It is best to use only ASCII characters for user names and passwords.
- The server uses MySQL passwords stored in the `user` table to authenticate client connections using MySQL built-in authentication. These passwords have nothing to do with passwords for logging in to your operating system. There is no necessary connection between the “external” password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.
- MySQL encrypts passwords stored in the `user` table using its own algorithm. This encryption is the same as that implemented by the `PASSWORD()` [868] SQL function but differs from that used during the Unix login process. Unix password encryption is the same as that implemented by the `ENCRYPT()` [867] SQL function. See the descriptions of the `PASSWORD()` [868] and `ENCRYPT()` [867] functions in Section 11.12, “Encryption and Compression Functions”.

  From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. (In earlier versions, even though passwords are stored in encrypted form in the `user` table, knowledge of the encrypted password value could be used to connect to the MySQL server.) Section 5.4.2.3, “Password Hashing in MySQL”, discusses password encryption further.

When you install MySQL, the grant tables are populated with an initial set of accounts. The names and access privileges for these accounts are described in Section 2.10.3, “Securing the Initial MySQL Accounts”, which also discusses how to assign passwords to them. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as `GRANT` and `REVOKE`. See Section 12.4.1, “Account Management Statements”.

When you connect to a MySQL server with a command-line client, specify the user name and password as necessary for the account that you want to use:

```
shell> mysql --user=monty --password=password db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u monty -ppassword db_name
```

There must be *no space* between the `-p` option and the following password value.

If you omit the *`password`* value following the `--password` [226] or `-p` option on the command line, the client prompts for one.

Specifying a password on the command line should be considered insecure. See Section 5.4.2.2, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

For additional information about specifying user names, passwords, and other connection parameters, see Section 4.2.2, “Connecting to the MySQL Server”.

5.6.2 Adding User Accounts

You can create MySQL accounts in two ways:

- By using `GRANT` statements. These statements cause the server to make appropriate modifications to the grant tables.
- By manipulating the MySQL grant tables directly with statements such as `INSERT`, `UPDATE`, or `DELETE`.

The preferred method is to use `GRANT` statements, because they are more concise and less error-prone than manipulating the grant tables directly. `GRANT` is described in Section 12.4.1.2, “`GRANT` Syntax”.

Another option for creating accounts is to use one of several available third-party programs that offer capabilities for MySQL account administration. `phpMyAdmin` is one such program.

The following examples show how to use the `mysql` client program to set up new accounts. These examples assume that privileges have been set up according to the defaults described in Section 2.10.3, “Securing the Initial MySQL Accounts”. This means that to make changes, you must connect to the MySQL server as the MySQL `root` user, and the `root` account must have the `INSERT` [492] privilege for the `mysql` database and the `RELOAD` [492] administrative privilege.

First, use the `mysql` program to connect to the server as the MySQL `root` user:

```
shell> mysql --user=root mysql
```

If you have assigned a password to the `root` account, you also need to supply a `--password` or `-p` option, both for this `mysql` command and for those later in this section.

After connecting to the server as `root`, you can add new accounts. The following statements use `GRANT` to set up four new accounts:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> GRANT USAGE ON *.* TO 'dummy'@'localhost';
```

The accounts created by these statements have the following properties:

- Two of the accounts have a user name of `monty` and a password of `some_pass`. Both accounts are superuser accounts with full privileges to do anything. The `'monty'@'localhost'` account can be used only when connecting from the local host. The `'monty'@'%'` account uses the `'%'` wildcard for the host part, so it can be used to connect from any host.

  It is necessary to have both accounts for `monty` to be able to connect from anywhere as `monty`. Without the `localhost` account, the anonymous-user account for `localhost` that is created by `mysql_install_db` would take precedence when `monty` connects from the local host. As a result, `monty` would be treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific `Host` column value than the `'monty'@'%'` account and thus comes earlier in the `user` table sort order. (`user` table sorting is discussed in Section 5.5.4, “Access Control, Stage 1: Connection Verification”.)

- The `'admin'@'localhost'` account has no password. This account can be used only by `admin` to connect from the local host. It is granted the `RELOAD` [492] and `PROCESS` [492] administrative privileges. These privileges enable the `admin` user to execute the `mysqladmin reload`, `mysqladmin refresh`, and `mysqladmin flush-xxx` commands, as well as `mysqladmin processlist` . No

privileges are granted for accessing any databases. You could add such privileges later by issuing other GRANT statements.

- The 'dummy'@'localhost' account has no password. This account can be used only to connect from the local host. No privileges are granted. The USAGE [493] privilege in the GRANT statement enables you to create an account without giving it any privileges. It has the effect of setting all the global privileges to 'N'. It is assumed that you will grant specific privileges to the account later.

To check the privileges for an account, use SHOW GRANTS:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----------------------------------------------------+
| Grants for admin@localhost                          |
+-----------------------------------------------------+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------------------+
```

As an alternative to GRANT, you can create the same accounts directly by issuing INSERT statements and then telling the server to reload the grant tables:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user
    ->     VALUES('localhost','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
    ->     VALUES('%','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
    ->     Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

When you create accounts with INSERT, it is necessary to use FLUSH PRIVILEGES to tell the server to reload the grant tables. Otherwise, the changes go unnoticed until you restart the server. With GRANT, FLUSH PRIVILEGES is unnecessary.

The reason for using the PASSWORD() [868] function with INSERT is to encrypt the password. The GRANT statement encrypts the password for you, so PASSWORD() [868] is unnecessary.

The 'Y' values enable privileges for the accounts. Depending on your MySQL version, you may have to use a different number of 'Y' values in the first two INSERT statements. (Versions prior to 3.22.11 have fewer privilege columns, and versions from 4.0.2 on have more.) The INSERT statement for the admin account employs the more readable extended INSERT syntax using SET that is available starting with MySQL 3.22.11 is used.

In the INSERT statement for the dummy account, only the Host, User, and Password columns in the user table row are assigned values. None of the privilege columns are set explicitly, so MySQL assigns them all the default value of 'N'. This is equivalent to what GRANT USAGE does.

To set up a superuser account, it is necessary only to insert a user table row with all privilege columns set to 'Y'. The user table privileges are global, so no entries in any of the other grant tables are needed.

The next examples create three accounts and give them access to specific databases. Each of them has a user name of custom and password of obscure.

To create the accounts with GRANT, use the following statements:

```
shell> mysql --user=root mysql
```

```
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON bankaccount.*
    ->     TO 'custom'@'localhost'
    ->     IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON expenses.*
    ->     TO 'custom'@'host47.example.com'
    ->     IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON customer.*
    ->     TO 'custom'@'server.domain'
    ->     IDENTIFIED BY 'obscure';
```

The three accounts can be used as follows:

- The first account can access the `bankaccount` database, but only from the local host.
- The second account can access the `expenses` database, but only from the host `host47.example.com`.
- The third account can access the `customer` database, but only from the host `server.domain`.

To set up the `custom` accounts without `GRANT`, use `INSERT` statements as follows to modify the grant tables directly:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('host47.example.com','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('server.domain','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('localhost','bankaccount','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('host47.example.com','expenses','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('server.domain','customer','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

The first three `INSERT` statements add `user` table entries that permit the user `custom` to connect from the various hosts with the given password, but grant no global privileges (all privileges are set to the default value of `'N'`). The next three `INSERT` statements add `db` table entries that grant privileges to `custom` for the `bankaccount`, `expenses`, and `customer` databases, but only when accessed from the proper hosts. As usual when you modify the grant tables directly, you must tell the server to reload them with `FLUSH PRIVILEGES` so that the privilege changes take effect.

To create a user who has access from all machines in a given domain (for example, `mydomain.com`), you can use the "`%`" wildcard character in the host part of the account name:

```
mysql> GRANT ...
    ->     ON *.*
    ->     TO 'myname'@'%.mydomain.com'
    ->     IDENTIFIED BY 'mypass';
```

To do the same thing by modifying the grant tables directly, do this:

```
mysql> INSERT INTO user (Host,User,Password,...)
    ->     VALUES('%.mydomain.com','myname',PASSWORD('mypass'),...);
mysql> FLUSH PRIVILEGES;
```

5.6.3 Removing User Accounts

To remove an account, use the `DROP USER` statement, which was added in MySQL 4.1.1. For older versions of MySQL, use `DELETE` instead. The account removal procedure is described in Section 12.4.1.1, “`DROP USER` Syntax”.

5.6.4 Setting Account Resource Limits

Before MySQL 4.0.2, the only means of limiting use of MySQL server resources is to set the global `max_user_connections` [420] system variable to a nonzero value. This limits the number of simultaneous connections that can be made by any given account, but places no limits on what a client can do once connected. In addition, this method is strictly global, and does not enable management of individual accounts. Both types of control are of interest to many MySQL administrators, particularly those working for Internet Service Providers.

Starting from MySQL 4.0.2, you can limit access to the following server resources for individual accounts:

- The number of queries that an account can issue per hour
- The number of updates that an account can issue per hour
- The number of times an account can connect to the server per hour

Any statement that a client can issue counts against the query limit (unless its results are served from the query cache). Only statements that modify databases or tables count against the update limit.

An “account” in this context is assessed against the actual host from which a user connects. Suppose that there is a row in the `user` table that has `User` and `Host` values of `usera` and `%.example.com`, to permit `usera` to connect from any host in the `example.com` domain. If `usera` connects simultaneously from `host1.example.com` and `host2.example.com`, the server applies the account resource limits separately to each connection. If `usera` connects again from `host1.example.com`, the server applies the limits for that connection together with the existing connection from that host.

The server limits account resources based on the resource-related columns of the `user` table in the `mysql` database: `max_questions`, `max_updates`, `max_connections`, and `max_user_connections`. If your `user` table does not have these columns, it must be upgraded; see Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

To set resource limits for an account, use the `GRANT` statement (see Section 12.4.1.2, “`GRANT` Syntax”). Provide a `WITH` clause that names each resource to be limited. For example, to create a new account that can access the `customer` database, but only in a limited fashion, issue this statement:

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     IDENTIFIED BY 'frank'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5;
```

The limit types need not all be named in the `WITH` clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. If the `GRANT` statement has no `WITH` clause, the limits are each set to the default value of zero (that is, no limit).

To modify existing limits for an account, use a `GRANT USAGE` statement at the global level (`ON *.*`). The following statement changes the query limit for `francis` to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

The statement modifies only the limit value specified and leaves the account otherwise unchanged.

To remove a limit, set its value to zero. For example, to remove the limit on how many times per hour `francis` can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

The server stores resource limits for an account in the `user` table row corresponding to the account. The `max_questions`, `max_updates`, and `max_connections` columns store the per-hour limits. (See Section 5.5.2, “Privilege System Grant Tables”.)

Resource-use counting takes place when any account has a nonzero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, further connections for the account are rejected until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, further queries or updates are rejected until the hour is up. In all such cases, an appropriate error message is issued.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a `mysqladmin reload` command).
- The counts for an individual account can be set to zero by re-granting it any of its limits. To do this, use `GRANT USAGE` as described earlier and specify a limit value equal to the value that the account currently has.

All counts begin at zero when the server starts; counts are not carried over through a restart.

5.6.5 Assigning Account Passwords

Required credentials for clients that connect to the MySQL server can include a password. This section describes how to assign passwords for MySQL accounts.

To assign a password when you create a new account with `GRANT`, include an `IDENTIFIED BY` clause:

```
mysql> GRANT ALL ON db.* TO 'jeffrey'@'localhost'
    -> IDENTIFIED BY 'mypass';
```

To assign or change a password for an existing account, one way is to issue a `SET PASSWORD` statement:

```
mysql> SET PASSWORD FOR
    -> 'jeffrey'@'localhost' = PASSWORD('mypass');
```

MySQL stores passwords in the `user` table in the `mysql` database. Only users such as `root` that have update access to the `mysql` database can change the password for other users. If you are not connected as an anonymous user, you can change your own password by omitting the `FOR` clause:

```
mysql> SET PASSWORD = PASSWORD('mypass');
```

You can also use a `GRANT USAGE` statement at the global level (`ON *.*`) to assign a password to an account without affecting the account's current privileges:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'localhost'
    -> IDENTIFIED BY 'mypass';
```

To assign a password from the command line, use the `mysqladmin` command:

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

The account for which this command sets the password is the one with a `user` table row that matches *`user_name`* in the `User` column and the client host *from which you connect* in the `Host` column.

It is preferable to assign passwords using one of the preceding methods, but it is also possible to modify the `user` table directly. In this case, you must also use `FLUSH PRIVILEGES` to cause the server to reread the grant tables. Otherwise, the change remains unnoticed by the server until you restart it.

- To establish a password for a new account, provide a value for the `Password` column:

  ```
  mysql> INSERT INTO mysql.user (Host,User,Password)
      -> VALUES('localhost','jeffrey',PASSWORD('mypass'));
  mysql> FLUSH PRIVILEGES;
  ```

- To change the password for an existing account, use `UPDATE` to set the `Password` column value:

  ```
  mysql> UPDATE mysql.user SET Password = PASSWORD('bagel')
      -> WHERE Host = 'localhost' AND User = 'francis';
  mysql> FLUSH PRIVILEGES;
  ```

During authentication when a client connects to the server, MySQL treats the password in the `user` table as an encrypted hash value (the value that `PASSWORD()` [868] would return for the password). When assigning a password to an account, it is important to store an encrypted value, not the plaintext password. Use the following guidelines:

- When you assign a password using `GRANT` with an `IDENTIFIED BY` clause or with the `mysqladmin password` command, they encrypt the password for you. Specify the literal plaintext password:

  ```
  mysql> GRANT USAGE ON *.* TO 'jeffrey'@'localhost'
      -> IDENTIFIED BY 'mypass';
  ```

- For `GRANT`, you can avoid sending the plaintext password if you know the hashed value that `PASSWORD()` [868] would return for the password. Specify the hashed value preceded by the keyword `PASSWORD`:

  ```
  mysql> GRANT USAGE ON *.* TO 'jeffrey'@'localhost'
      -> IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
  ```

- When you assign an account a nonempty password using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must use the `PASSWORD()` [868] function to encrypt the password, otherwise the password is stored as plaintext. Suppose that you assign a password like this:

```
mysql> SET PASSWORD FOR
    -> 'jeffrey'@'localhost' = 'mypass';
```

The result is that the literal value `'mypass'` is stored as the password in the `user` table, not the encrypted value. When `jeffrey` attempts to connect to the server using this password, the value is encrypted and compared to the value stored in the `user` table. However, the stored value is the literal string `'mypass'`, so the comparison fails and the server rejects the connection with an `Access denied` error.

Note

`PASSWORD()` [868] encryption differs from Unix password encryption. See Section 5.6.1, “User Names and Passwords”.

5.6.6 Using SSL for Secure Connections

Beginning with version 4.0.0, MySQL has support for secure (encrypted) connections between MySQL clients and the server using the Secure Sockets Layer (SSL) protocol. This section discusses how to use SSL connections. For information on how to require users to use SSL connections, see the discussion of the `REQUIRE` clause of the `GRANT` statement in Section 12.4.1.2, “`GRANT` Syntax”.

The standard configuration of MySQL is intended to be as fast as possible, so encrypted connections are not used by default. Doing so would make the client/server protocol much slower. Encrypting data is a CPU-intensive operation that requires the computer to do additional work and can delay other MySQL tasks. For applications that require the security provided by encrypted connections, the extra computation is warranted.

MySQL enables encryption on a per-connection basis. You can choose a normal unencrypted connection or a secure encrypted SSL connection according the requirements of individual applications.

Secure connections are based on the OpenSSL API and are available through the MySQL C API. Replication uses the C API, so secure connections can be used between master and slave servers.

Another way to connect securely is from within an SSH connection to the MySQL server host. For an example, see Section 5.6.7, “Connecting to MySQL Remotely from Windows with SSH”.

5.6.6.1 Basic SSL Concepts

To understand how MySQL uses SSL, it is necessary to explain some basic SSL and X509 concepts. People who are familiar with these can skip this part of the discussion.

By default, MySQL uses unencrypted connections between the client and the server. This means that someone with access to the network could watch all your traffic and look at the data being sent or received. They could even change the data while it is in transit between client and server. To improve security a little, you can compress client/server traffic by using the `--compress` option when invoking client programs. However, this does not foil a determined attacker.

When you need to move information over a network in a secure fashion, an unencrypted connection is unacceptable. Encryption is the way to make any kind of data unreadable. In fact, today's practice requires many additional security elements from encryption algorithms. They should resist many kind of known attacks such as changing the order of encrypted messages or replaying data twice.

SSL is a protocol that uses different encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect any data change, loss, or replay. SSL also incorporates algorithms that provide identity verification using the X509 standard.

X509 makes it possible to identify someone on the Internet. It is most commonly used in e-commerce applications. In basic terms, there should be some company called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

If you need more information about SSL, X509, or encryption, use your favorite Internet search engine to search for the keywords in which you are interested.

5.6.6.2 Using SSL Connections

To use SSL connections between the MySQL server and client programs, your system must support OpenSSL and your version of MySQL must be 4.0.0 or newer and built with SSL support.

To get secure connections to work with MySQL and SSL, you must do the following:

1. Install the OpenSSL library if it has not already been installed. We have tested MySQL with OpenSSL 0.9.6. To obtain OpenSSL, visit http://www.openssl.org.

   Building MySQL using OpenSSL requires a shared OpenSSL library, otherwise linker errors occur.

2. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, configure a MySQL source distribution to use SSL. When you configure MySQL, invoke the `configure` script with the `--with-vio` and `--with-openssl` options:

   ```
   shell> ./configure --with-vio --with-openssl
   ```

3. Make sure that the `user` in the `mysql` database includes the SSL-related columns (beginning with `ssl_` and `x509_`). If your `user` table does not have these columns, it must be upgraded; see Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

4. To check whether a server binary is compiled with SSL support, invoke it with the `--ssl` [521] option. An error will occur if the server does not support SSL:

   ```
   shell> mysqld --ssl --help
   060525 14:18:52 [ERROR] mysqld: unknown option '--ssl'
   ```

   To check whether a running `mysqld` server supports SSL, examine the value of the `have_openssl` [413] system variable:

   ```
   mysql> SHOW VARIABLES LIKE 'have_openssl';
   +---------------+-------+
   | Variable_name | Value |
   +---------------+-------+
   | have_openssl  | YES   |
   +---------------+-------+
   ```

   If the value is `YES`, the server supports OpenSSL connections.

To enable SSL connections, the proper SSL-related options must be used (see Section 5.6.6.3, “SSL Command Options”).

To start the MySQL server so that it permits clients to connect using SSL, use the options that identify the key and certificate files the server needs when establishing a secure connection:

```
shell> mysqld --ssl-ca=ca-cert.pem \
       --ssl-cert=server-cert.pem \
       --ssl-key=server-key.pem
```

- `--ssl-ca` [522] identifies the Certificate Authority (CA) certificate.
- `--ssl-cert` [522] identifies the server public key. This can be sent to the client and authenticated against the CA certificate that it has.
- `--ssl-key` [522] identifies the server private key.

To establish a secure connection to a MySQL server with SSL support, the options that a client must specify depend on the SSL requirements of the user account that the client uses. (See the discussion of the `REQUIRE` clause in Section 12.4.1.2, “`GRANT` Syntax”.)

If the account has no special SSL requirements or was created using a `GRANT` statement that includes the `REQUIRE SSL` option, a client can connect securely by using just the `--ssl-ca` [522] option:

```
shell> mysql --ssl-ca=ca-cert.pem
```

To require that a client certificate also be specified, create the account using the `REQUIRE X509` option. Then the client must also specify the proper client key and certificate files or the server will reject the connection:

```
shell> mysql --ssl-ca=ca-cert.pem \
       --ssl-cert=client-cert.pem \
       --ssl-key=client-key.pem
```

In other words, the options are similar to those used for the server. Note that the Certificate Authority certificate has to be the same.

A client can determine whether the current connection with the server uses SSL by checking the value of the `Ssl_cipher` [455] status variable. The value of `Ssl_cipher` [455] is nonempty if SSL is used, and empty otherwise. For example:

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

For the `mysql` client, you can use the `STATUS` or `\s` command and check the `SSL` line:

```
mysql> \s
...
SSL:                    Not in use
...
```

Or:

```
mysql> \s
...
SSL:                    Cipher in use is DHE-RSA-AES256-SHA
...
```

To establish a secure connection from within an application program, use the `mysql_ssl_set()` C API function to set the appropriate certificate options before calling `mysql_real_connect()`. See Section 17.6.6.65, “`mysql_ssl_set()`”.

5.6.6.3 SSL Command Options

The following list describes options that are used for specifying the use of SSL, certificate files, and key files. These options are available beginning with MySQL 4.0. They can be given on the command line or in an option file. These options are not available unless MySQL has been built with SSL support. See Section 5.6.6.2, “Using SSL Connections”. (There are also `--master-ssl*` options that can be used for setting up a secure connection from a slave replication server to a master server; see Section 14.8, “Replication and Binary Logging Options and Variables”.)

Table 5.11 SSL Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
have_openssl [413]			Yes		Global	No
skip-ssl [521]	Yes	Yes				
ssl [521]	Yes	Yes				
ssl-ca [522]	Yes	Yes			Global	No
- *Variable*: ssl_ca			Yes		Global	No
ssl-capath [522]	Yes	Yes			Global	No
- *Variable*: ssl_capath			Yes		Global	No
ssl-cert [522]	Yes	Yes			Global	No
- *Variable*: ssl_cert			Yes		Global	No
ssl-cipher [522]	Yes	Yes			Global	No
- *Variable*: ssl_cipher			Yes		Global	No
ssl-key [522]	Yes	Yes			Global	No
- *Variable*: ssl_key			Yes		Global	No

- `--ssl` [521]

  For the server, this option specifies that the server permits SSL connections. For a client program, it permits the client to connect to the server using SSL. This option is not sufficient in itself to cause an SSL connection to be used. You must also specify the `--ssl-ca` [522] option, and possibly the `--ssl-cert` [522] and `--ssl-key` [522] options.

  This option is more often used in its opposite form to override any other SSL options and indicate that SSL should *not* be used. To do this, specify the option as `--skip-ssl` [521] or `--ssl=0` [521].

  Note that use of `--ssl` [521] does not *require* an SSL connection. For example, if the server or client is compiled without SSL support, a normal unencrypted connection is used.

  The secure way to require use of an SSL connection is to create an account on the server that includes a `REQUIRE SSL` clause in the `GRANT` statement. Then use that account to connect to the server, where both the server and the client have SSL support enabled.

  The `REQUIRE` clause permits other SSL-related restrictions as well. The description of `REQUIRE` in Section 12.4.1.2, “`GRANT` Syntax”, provides additional detail about which SSL command options may or must be specified by clients that connect using accounts that are created using the various `REQUIRE` options.

- `--ssl-ca=file_name` [522]

  The path to a file that contains a list of trusted SSL CAs.

- `--ssl-capath=directory_name` [522]

  The path to a directory that contains trusted SSL CA certificates in PEM format.

- `--ssl-cert=file_name` [522]

  The name of the SSL certificate file to use for establishing a secure connection.

- `--ssl-cipher=cipher_list` [522]

  A list of permissible ciphers to use for SSL encryption. For greatest portability, `cipher_list` should be a list of one or more cipher names, separated by colons. Examples:

```
--ssl-cipher=AES128-SHA
--ssl-cipher=DHE-RSA-AES256-SHA:AES128-SHA
```

  This format is understood both by OpenSSL and yaSSL. OpenSSL supports a more flexible syntax for specifying ciphers, as described in the OpenSSL documentation at http://www.openssl.org/docs/apps/ciphers.html. However, this extended syntax will fail if used with a MySQL installation compiled against yaSSL (which may be the case for MySQL 5.0 and up).

  If no cipher in the list is supported, SSL connections will not work.

- `--ssl-key=file_name` [522]

  The name of the SSL key file to use for establishing a secure connection.

5.6.6.4 Setting Up SSL Certificates for MySQL

This section demonstrates how to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. The first two examples are intended for use on Unix and both use the `openssl` command that is part of OpenSSL. The third example describes how to set up SSL files on Windows.

Following the third example, instructions are given for using the files to test SSL connections. You can also use the files as described in Section 5.6.6.2, “Using SSL Connections”.

Example 1: Creating SSL files from the command line on Unix

The following example shows a set of commands to create MySQL server and client certificate and key files. You will need to respond to several prompts by the `openssl` commands. For testing, you can press Enter to all prompts. For production use, you should provide nonempty responses.

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts

# Create CA certificate
shell> openssl genrsa 2048 > ca-key.pem
shell> openssl req -new -x509 -nodes -days 1000 \
         -key ca-key.pem > ca-cert.pem

# Create server certificate
shell> openssl req -newkey rsa:2048 -days 1000 \
```

```
        -nodes -keyout server-key.pem > server-req.pem
shell> openssl x509 -req -in server-req.pem -days 1000 \
        -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > server-cert.pem

# Create client certificate
shell> openssl req -newkey rsa:2048 -days 1000 \
        -nodes -keyout client-key.pem > client-req.pem
shell> openssl x509 -req -in client-req.pem -days 1000 \
        -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 > client-cert.pem
```

Example 2: Creating SSL files using a script on Unix

Here is an example script that shows how to set up SSL certificates for MySQL:

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca-cert.pem \
    -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ................++++++
# .........++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
```

```
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# ..........++++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca  -policy policy_anything -out $DIR/server-cert.pem \
    -config $DIR/openssl.cnf -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
```

```
# .....................................++++++
# ..........................................++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca  -policy policy_anything -out $DIR/client-cert.pem \
    -config $DIR/openssl.cnf -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cnf=""
cnf="$cnf [client]"
cnf="$cnf ssl-ca=$DIR/ca-cert.pem"
cnf="$cnf ssl-cert=$DIR/client-cert.pem"
cnf="$cnf ssl-key=$DIR/client-key.pem"
cnf="$cnf [mysqld]"
```

```
cnf="$cnf ssl-ca=$DIR/ca-cert.pem"
cnf="$cnf ssl-cert=$DIR/server-cert.pem"
cnf="$cnf ssl-key=$DIR/server-key.pem"
echo $cnf | replace " " '
' > $DIR/my.cnf
```

Example 3: Creating SSL files on Windows

Download OpenSSL for Windows. An overview of available packages can be seen here: http://www.slproweb.com/products/Win32OpenSSL.html

Choose of the following packages, depending on your architecture (32-bit or 64-bit):

- Win32 OpenSSL v0.9.8l Light, available at: http://www.slproweb.com/download/Win32OpenSSL_Light-0_9_8l.exe
- Win64 OpenSSL v0.9.8l Light, available at: http://www.slproweb.com/download/Win64OpenSSL_Light-0_9_8l.exe

if a message occurs during setup indicating `'...critical component is missing: Microsoft Visual C++ 2008 Redistributables'`, cancel the setup and download one of the following packages as well, again depending on your architecture (32-bit or 64-bit):

- Visual C++ 2008 Redistributables (x86), available at: http://www.microsoft.com/downloads/details.aspx?familyid=9B2DA534-3E03-4391-8A4D-074B9F2BC1BF“isplaylang=en
- Visual C++ 2008 Redistributables (x64), available at: http://www.microsoft.com/downloads/details.aspx?familyid=bd2a6171-e2d6-4230-b809-9a8d7548c1b6“isplaylang=en

After installing the additional package, restart the OpenSSL setup.

During installation, leave the default `C:\OpenSSL` as the install path, and also leave the default option `'Copy OpenSSL DLL files to the Windows system directory'` selected.

When the installation has finished, add `C:\OpenSSL\bin` to the Windows System Path variable of your server:

1. On the Windows desktop, right-click the My Computer icon, and select Properties.
2. Next select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.
3. Under **System Variables**, select Path, and then click the Edit button. The Edit System Variable dialogue should appear.
4. Add `';C:\OpenSSL\bin'` to the end (notice the semicolon).
5. Press OK 3 times.
6. Check that OpenSSL was correctly integrated into the Path variable by opening a new command console (`Start>Run>cmd.exe`) and verifying that OpenSSL is available:

```
Microsoft Windows [Version ...]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd \

C:\>openssl
OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.

C:\>
```

Depending on your version of Windows, the preceding instructions might be slightly different.

After OpenSSL has been installed, use the instructions from Example 1 (shown earlier in this section), with the following changes:

- Change the follow Unix commands:

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts
```

  On Windows, use these commands instead:

```
# Create clean environment
shell> md c:\newcerts
shell> cd c:\newcerts
```

- When a `'\'` character is shown at the end of a command line, this `'\'` character must be removed and the command lines entered all on a single line.
- For references to `my.cnf` option files, substitute `my.ini` instead.

Testing SSL connections

To test SSL connections, start the server as follows, where `$DIR` is the path name to the directory where the sample `my.cnf` option file is located:

```
shell> mysqld --defaults-file=$DIR/my.cnf &
```

Then invoke a client program using the same option file:

```
shell> mysql --defaults-file=$DIR/my.cnf
```

If you have a MySQL source distribution, you can also test your setup by modifying the preceding `my.cnf` file to refer to the demonstration certificate and key files in the `SSL` directory of the distribution.

5.6.7 Connecting to MySQL Remotely from Windows with SSH

This section describes how to get a secure connection to a remote MySQL server with SSH. The information was provided by David Carlson `<dcarlson@mplcomm.com>`.

1. Install an SSH client on your Windows machine. As a user, the best nonfree one I have found is from `SecureCRT` from http://www.vandyke.com/. Another option is `f-secure` from http://www.f-secure.com/. You can also find some free ones on `Google` at http://directory.google.com/Top/Computers/Internet/Protocols/SSH/Clients/Windows/.
2. Start your Windows SSH client. Set `Host_Name = yourmysqlserver_URL_or_IP`. Set `userid=your_userid` to log in to your server. This `userid` value might not be the same as the user name of your MySQL account.
3. Set up port forwarding. Either do a remote forward (Set `local_port: 3306`, `remote_host: yourmysqlservername_or_ip`, `remote_port: 3306`) or a local forward (Set `port: 3306`, `host: localhost`, `remote port: 3306`).
4. Save everything, otherwise you will have to redo it the next time.
5. Log in to your server with the SSH session you just created.

6. On your Windows machine, start some ODBC application (such as Access).
7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not *`yourmysqlservername`*.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

5.6.8 Auditing MySQL Account Activity

Applications can use the following guidelines to perform auditing that ties database activity to MySQL accounts.

MySQL accounts correspond to rows in the `mysql.user` table. When a client connects successfully, the server authenticates the client to a particular row in this table. The `User` and `Host` column values in this row uniquely identify the account and correspond to the *`'user_name'@'host_name'`* format in which account names are written in SQL statements.

The account used to authenticate a client determines which privileges the client has. Normally, the `CURRENT_USER()` [872] function can be invoked to determine which account this is for the client user. Its value is constructed from the `User` and `Host` columns of the `user` table row for the account.

To determine the invoking user, you can also call the `USER()` [876] function, which returns a value indicating the actual user name provided by the client and the host from which the client connected. However, this value does not necessarily correspond directly to an account in the `user` table, because the `USER()` [876] value never contains wildcards, whereas account values (as returned by `CURRENT_USER()` [872]) may contain user name and host name wildcards.

For example, a blank user name matches any user, so an account of `''@'localhost'` enables clients to connect as an anonymous user from the local host with any user name. If this case, if a client connects as `user1` from the local host, `USER()` [876] and `CURRENT_USER()` [872] return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+----------------+----------------+
| USER()         | CURRENT_USER() |
+----------------+----------------+
| user1@localhost | @localhost     |
+----------------+----------------+
```

The host name part of an account can contain wildcards, too. If the host name contains a `'%'` or `'_'` pattern character or uses netmask notation, the account can be used for clients connecting from multiple hosts and the `CURRENT_USER()` [872] value will not indicate which one. For example, the account `'user2'@'%.example.com'` can be used by `user2` to connect from any host in the `example.com` domain. If `user2` connects from `remote.example.com`, `USER()` [876] and `CURRENT_USER()` [872] return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+-------------------------+---------------------+
| USER()                  | CURRENT_USER()      |
+-------------------------+---------------------+
| user2@remote.example.com | user2@%.example.com |
+-------------------------+---------------------+
```

If an application invokes `USER()` [876] for user auditing, but must also be able to associate the `USER()` [876] value with an account in the `user` table, it is necessary to avoid accounts that contain wildcards in the `User` or `Host` column. Specifically, do not permit `User` to be empty (which creates an anonymous-user account), and do not permit pattern characters or netmask notation in `Host` values. All accounts must have a nonempty `User` value and literal `Host` value.

With respect to the previous examples, the `''@'localhost'` and `'user2'@'%.example.com'` accounts should be changed not to use wildcards:

```
RENAME USER ''@'localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%.example.com' TO 'user2'@'remote.example.com';
```

If `user2` must be able to connect from several hosts in the `example.com` domain, there should be a separate account for each host.

To extract the user name or host name part from a `CURRENT_USER()` [872] or `USER()` [876] value, use the `SUBSTRING()` [802] function:

```
mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',1);
+--------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',1) |
+--------------------------------------+
| user1                                |
+--------------------------------------+

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',-1);
+---------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',-1) |
+---------------------------------------+
| localhost                             |
+---------------------------------------+
```

5.7 Running Multiple MySQL Servers on the Same Machine

In some cases, you might want to run multiple `mysqld` servers on the same machine. You might want to test a new MySQL release while leaving your existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

To run multiple servers on a single machine, each server must have unique values for several operating parameters. These can be set on the command line or in option files. See Section 4.2.3, “Specifying Program Options”.

At least the following options must be different for each server:

- `--port=`*`port_num`* [391]

  `--port` [391] controls the port number for TCP/IP connections. (Alternatively, if the host has multiple network addresses, you can use `--bind-address` [384] to cause different servers to listen to different interfaces.)

- `--socket=`*`path`* [394]

  `--socket` [394] controls the Unix socket file path on Unix and the name of the named pipe on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers that support named-pipe connections.

- `--shared-memory-base-name=`*`name`* [392]

  The name of shared memory to use for shared-memory connections. This option is available only on Windows. The default name is `MYSQL`. The name is case sensitive. This option was added in MySQL 4.1.

- `--pid-file=file_name` [391]

  This option is used only on Unix. It indicates the path name of the file in which the server writes its process ID.

If you use the following log file options, they must be different for each server:

- `--log[=file_name]` [388]
- `--log-bin[=file_name]` [1181]
- `--log-update[=file_name]` [389]
- `--log-error[=file_name]` [388]
- `--log-isam=file_name` [388]
- `--bdb-logdir=file_name` [1138]

Section 5.3.6, “Server Log Maintenance”, discusses the log file options further.

For better performance, you can specify the following options differently for each server, to spread the load between several physical disks:

- `--tmpdir=path` [394]
- `--bdb-tmpdir=path` [1138]

Having different temporary directories also makes it easier to determine which MySQL server created any given temporary file.

With very limited exceptions, each server should use a different data directory, which is specified using the `--datadir=path` [385] option.

Warning

Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log file names that are unique to each server. Otherwise, the servers try to log to the same files. Please note that this kind of setup only works with `ISAM`, `MyISAM` and `MERGE` tables, and not with any of the other storage engines.

The warning against sharing a data directory among servers also applies in an NFS environment. Permitting multiple MySQL servers to access a common data directory over NFS is a *very bad idea*.

- The primary problem is that NFS is the speed bottleneck. It is not meant for such use.
- Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Make it easy for yourself: Forget about sharing a data directory among servers over NFS. A better solution is to have one computer that contains several CPUs and use an operating system that handles threads efficiently.

If you have multiple MySQL installations in different locations, you can specify the base installation directory for each server with the `--basedir=path` [384] option to cause each server to use a different

data directory, log files, and PID file. (The defaults for all these values are determined relative to the base directory). In that case, the only other options you need to specify are the `--socket` [394] and `--port` [391] options. Suppose that you install different versions of MySQL using tarfile binary distributions. These install in different locations, so you can start the server for each installation using `bin/mysqld_safe` under its own corresponding base directory. `mysqld_safe` determines the proper `--basedir` [384] option to pass to `mysqld`, and you need specify only the `--socket` [245] and `--port` [245] options to `mysqld_safe`. (For versions of MySQL older than 4.0, use `safe_mysqld` rather than `mysqld_safe`.)

As discussed in the following sections, it is possible to start additional servers by setting environment variables or by specifying appropriate command-line options. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` [235] option is useful for this purpose.

5.7.1 Running Multiple Servers on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters. On Windows NT-based systems, you also have the option of installing several servers as Windows services and running them that way. General instructions for running MySQL servers from the command line or as services are given in Section 2.3, “Installing MySQL on Microsoft Windows”. This section describes how to make sure that you start each server with different values for those startup options that must be unique per server, such as the data directory. These options are described in Section 5.7, “Running Multiple MySQL Servers on the Same Machine”.

5.7.1.1 Starting Multiple Windows Servers at the Command Line

To start multiple servers manually from the command line, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the file name with a `--defaults-file` [235] option when you run it.

Suppose that you want to run `mysqld` on port 3307 with a data directory of `C:\mydata1`, and `mysqld-max` on port 3308 with a data directory of `C:\mydata2`. (To do this, make sure that before you start the servers, each data directory exists and has its own copy of the `mysql` database that contains the grant tables.) Then create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

Create a second file named `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

Then start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-max --defaults-file=C:\my-opts2.cnf
```

On NT, each server starts in the foreground (no new prompt appears until the server exits later), so you will need to issue those two commands in separate console windows.

To shut down the servers, you must connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

Servers configured as just described permit clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to permit named-pipe connections, use the `mysqld-nt` or `mysqld-max-nt` servers and specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Then start the server this way:

```
C:\> C:\mysql\bin\mysqld-nt --defaults-file=C:\my-opts1.cnf
```

Modify `C:\my-opts2.cnf` similarly for use by the second server.

A similar procedure applies for servers that you want to support shared-memory connections. Enable such connections with the `--shared-memory` [392] option and specify a unique shared-memory name for each server with the `--shared-memory-base-name` [392] option.

5.7.1.2 Starting Multiple Windows Servers as Services

On NT-based systems, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in Section 2.3.11, “Starting MySQL as a Windows Service”.

As of MySQL 4.0.2, you can install multiple servers as services. In this case, you must make sure that each server uses a different service name in addition to all the other parameters that must be unique per server.

For the following instructions, assume that you want to run the `mysqld-nt` server from two different versions of MySQL that are installed at `C:\mysql-4.0.8` and `C:\mysql-4.0.17`, respectively. (This might be the case if you are running 4.0.8 as your production server, but want to test 4.0.17 before upgrading to it.)

The following principles apply when installing a MySQL service with the `--install` or `--install-manual` option:

- If you specify no service name, the server uses the default service name of `MySQL` and the server reads options from the `[mysqld]` group in the standard option files.
- If you specify a service name after the `--install` option, the server ignores the `[mysqld]` option group and instead reads options from the group that has the same name as the service. The server reads options from the standard option files.
- If you specify a `--defaults-file` [235] option after the service name, the server ignores the standard option files and reads options only from the `[mysqld]` group of the named file.

Note

Before MySQL 4.0.17, only a server installed using the default service name (`MySQL`) or one installed explicitly with a service name of `mysqld` will read the

> `[mysqld]` group in the standard option files. As of 4.0.17, all servers read the `[mysqld]` group if they read the standard option files, even if they are installed using another service name. This enables you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group named after each service for use by the server installed with that service name.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, be sure that you shut down and remove any existing MySQL services first.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 4.0.8 `mysqld-nt` using the service name of `mysqld1` and the 4.0.17 `mysqld-nt` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 4.0.8 and the `[mysqld2]` group for 4.0.17. For example, you can set up `C:\my.cnf` like this:

  ```
  # options for mysqld1 service
  [mysqld1]
  basedir = C:/mysql-4.0.8
  port = 3307
  enable-named-pipe
  socket = mypipe1

  # options for mysqld2 service
  [mysqld2]
  basedir = C:/mysql-4.0.17
  port = 3308
  enable-named-pipe
  socket = mypipe2
  ```

  Install the services as follows, using the full server path names to ensure that Windows registers the correct executable program for each service:

  ```
  C:\> C:\mysql-4.0.8\bin\mysqld-nt --install mysqld1
  C:\> C:\mysql-4.0.17\bin\mysqld-nt --install mysqld2
  ```

  To start the services, use the services manager, or use `NET START` with the appropriate service names:

  ```
  C:\> NET START mysqld1
  C:\> NET START mysqld2
  ```

  To stop the services, use the services manager, or use `NET STOP` with the appropriate service names:

  ```
  C:\> NET STOP mysqld1
  C:\> NET STOP mysqld2
  ```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` [235] when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

  With this approach, to specify options for the 4.0.8 `mysqld-nt`, create a file `C:\my-opts1.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-4.0.8
  port = 3307
  enable-named-pipe
  ```

```
socket = mypipe1
```

For the 4.0.17 `mysqld-nt`, create a file `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
basedir = C:/mysql-4.0.17
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows (enter each command on a single line):

```
C:\> C:\mysql-4.0.8\bin\mysqld-nt --install mysqld1
           --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-4.0.17\bin\mysqld-nt --install mysqld2
           --defaults-file=C:\my-opts2.cnf
```

To use a `--defaults-file` [235] option when you install a MySQL server as a service, you must precede the option with the service name.

After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use `mysqld --remove` for each one, specifying a service name following the `--remove` option. If the service name is the default (`MySQL`), you can omit it.

5.7.2 Running Multiple Servers on Unix

The easiest way is to run multiple servers on Unix is to compile them with different TCP/IP ports and Unix socket files so that each one is listening on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new server to have different operating parameters, use a `configure` command something like this:

```
shell> ./configure --with-tcp-port=port_number \
            --with-unix-socket-path=file_name \
            --prefix=/usr/local/mysql-4.0.17
```

Here, *`port_number`* and *`file_name`* must be different from the default TCP/IP port number and Unix socket file path name, and the `--prefix` [95] value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket file name:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

Note that if you specify `localhost` as a host name, `mysqladmin` defaults to using a Unix socket file connection rather than TCP/IP. In MySQL 4.1, you can explicitly specify the connection protocol to use by using the `--protocol={TCP|SOCKET|PIPE|MEMORY}` [226] option.

You do not have to compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` [394] and `--port` [391] option values, and pass a `--datadir=path` [385] option to `mysqld_safe` so that the server uses a different data directory.

Another way to achieve a similar effect is to use environment variables to set the Unix socket file name and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

Section 2.13, “Environment Variables”, includes a list of other environment variables you can use to affect `mysqld`.

For automatic server execution, the startup script that is executed at boot time should run the following command once for each server with an appropriate option file path for each command:

```
shell> mysqld_safe --defaults-file=file_name
```

Each option file should contain option values specific to a given server.

On Unix, the `mysqld_multi` script is another way to start multiple servers. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

5.7.3 Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name` [226] `--port=port_number` [226] to connect using TCP/IP to a remote server, with `--host=127.0.0.1` [226] `--port=port_number` [226] to connect using TCP/IP to a local server, or with `--host=localhost` [226] `--socket=file_name` [227] to connect to a local server using a Unix socket file or a Windows named pipe.

- As of MySQL 4.1, start the client with `--protocol=TCP` [226] to connect using TCP/IP, `--protocol=SOCKET` [226] to connect using a Unix socket file, `--protocol=PIPE` [226] to connect using a named pipe, or `--protocol=MEMORY` [226] to connect using shared memory. For TCP/IP connections, you may also need to specify `--host` [226] and `--port` [226] options. For the other types of connections, you may need to specify a `--socket` [227] option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` [226] option to specify the shared-memory name. Shared-memory connections are supported only on Windows.

- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific

socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Section 2.13, “Environment Variables”.

- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.2.3.3, “Using Option Files”.

- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling `mysql_options()`. See Section 17.6.6, “C API Function Descriptions”.

- If you are using the Perl `DBD::mysql` module, you can read options from MySQL option files. For example:

```
$dsn = "DBI:mysql:test;mysql_read_default_group=client;"
       . "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
$dbh = DBI->connect($dsn, $user, $password);
```

  See Section 17.8, “MySQL Perl API”.

  Other programming interfaces may provide similar capabilities for reading option files.

Chapter 6 Backup and Recovery

Table of Contents

It is important to back up your databases so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replication slave servers.

MySQL offers a variety of backup strategies from which you can choose the methods that best suit the requirements for your installation. This chapter discusses several backup and recovery topics with which you should be familiar:

- Types of backups: Logical versus physical, full versus incremental, and so forth
- Methods for creating backups
- Recovery methods, including point-in-time recovery
- Backup scheduling, compression, and encryption
- Table maintenance, to enable recovery of corrupt tables

Additional Resources

Resources related to backup or to maintaining data availability include the following:

- A forum dedicated to backup issues is available at http://forums.mysql.com/list.php?28.
- Details for `mysqldump`, `mysqlhotcopy`, and other MySQL backup programs can be found in Chapter 4, *MySQL Programs*.

- The syntax of the SQL statements described here is given in Chapter 12, *SQL Statement Syntax*.
- For additional information about `InnoDB` backup procedures, see Section 13.2.7, “Backing Up and Recovering an `InnoDB` Database”.
- Replication enables you to maintain identical data on multiple servers. This has several benefits, such as enabling client query load to be distributed over servers, availability of data even if a given server is taken offline or fails, and the ability to make backups with no impact on the master by using a slave server. See Chapter 14, *Replication*.
- MySQL Cluster provides a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. See Chapter 15, *MySQL Cluster*. For information specifically about MySQL Cluster backup, see Section 15.5.3, “Online Backup of MySQL Cluster”.

6.1 Backup and Recovery Types

This section describes the characteristics of different types of backups.

Logical Versus Physical (Raw) Backups

Logical backups save information represented as logical database structure (`CREATE DATABASE`, `CREATE TABLE` statements) and content (`INSERT` statements or delimited-text files). Physical backups consist of raw copies of the directories and files that store database contents.

Logical backup methods have these characteristics:

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running. The server is not taken offline.
- Logical backup tools include the `mysqldump` program and the `SELECT ... INTO OUTFILE` statement. These work for any storage engine, even `MEMORY`.
- To restore logical backups, SQL-format dump files can be processed using the `mysql` client. To load delimited-text files, use the `LOAD DATA INFILE` statement or the `mysqlimport` client.

Physical backup methods have these characteristics:

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory. Data from `MEMORY` tables cannot be backed up this way because their contents are not stored on disk.
- Physical backup methods are faster than logical because they involve only file copying without conversion.

- Output is more compact than for logical backup.
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine. (Each `MyISAM` table corresponds uniquely to a set of files, but an `InnoDB` table shares file storage with other `InnoDB` tables.)
- In addition to databases, the backup can include any related files such as log or configuration files.
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup.
- Physical backup tools include file system-level commands (such as `cp`, `scp`, `tar`, `rsync`), `mysqlhotcopy` for `MyISAM` tables, `ibbackup` for `InnoDB` tables, or `START BACKUP` for `NDB` tables.
- For restore, files copied at the file system level or with `mysqlhotcopy` can be copied back to their original locations with file system commands; `ibbackup` restores `InnoDB` tables, and `ndb_restore` restores `NDB` tables.

Online Versus Offline Backups

Online backups take place while the MySQL server is running so that the database information can be obtained from the server. Offline backups take place while the server is stopped. This distinction can also be described as “hot” versus “cold” backups; a “warm” backup is one where the server remains running but locked against modifying data while you access database files externally.

Online backup methods have these characteristics:

- The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
- Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity.

Offline backup methods have these characteristics:

- Clients can be affected adversely because the server is unavailable during backup.
- The backup procedure is simpler because there is no possibility of interference from client activity.

A similar distinction between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely that clients will be affected for online recovery than for online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

Local Versus Remote Backups

A local backup is performed on the same host where the MySQL server runs, whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server. host.

- `mysqldump` can connect to local or remote servers. For SQL output (`CREATE` and `INSERT` statements), local or remote dumps can be done and generate output on the client. For delimited-text output (with the `--tab` [299] option), data files are created on the server host.

- `mysqlhotcopy` performs only local backups: It connects to the server to lock it against data modifications and then copies local table files.
- `SELECT ... INTO OUTFILE` can be initiated from a local or remote client host, but the output file is created on the server host.
- Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

Snapshot Backups

Some file system implementations enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.) MySQL itself does not provide the capability for taking file system snapshots. It is available through third-party solutions such as Veritas, LVM, or ZFS.

Full Versus Incremental Backups

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section. Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

Full Versus Point-in-Time (Incremental) Recovery

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

Incremental recovery is recovery of changes made during a given time span. This is also called point-in-time recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

Table Maintenance

Data integrity can be compromised if tables become corrupt. MySQL provides programs for checking `MyISAM` and `ISAM` tables and repairing them should problems be found. See Section 6.6, “`MyISAM` Table Maintenance and Crash Recovery”.

Backup Scheduling, Compression, and Encryption

Backup scheduling is valuable for automating backup procedures. Compression of backup output reduces space requirements, and encryption of the output provides better security against unauthorized access of backed-up data. MySQL itself does not provide these capabilities. `ibbackup` can compress `InnoDB` backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

6.2 Database Backup Methods

This section summarizes some general methods for making backups.

Making Backups by Copying Table Files

For storage engines that represent each table using its own files, tables can be backed up by copying those files. For example, `MyISAM` tables are stored as files, so it is easy to do a backup by copying files (`*.frm`, `*.MYD`, and `*.MYI` files). To get a consistent backup, stop the server or do a `LOCK TABLES` on the relevant tables followed by `FLUSH TABLES` for the tables. See Section 12.3.5, "`LOCK TABLES` and `UNLOCK TABLES` Syntax", and Section 12.4.6.2, "`FLUSH` Syntax". You need only a read lock; this enables other clients to continue to query the tables while you are making a copy of the files in the database directory. The `FLUSH TABLES` statement is needed to ensure that the all active index pages are written to disk before you start the backup.

You can also create a binary backup simply by copying all table files, as long as the server isn't updating anything. The `mysqlhotcopy` script uses this method. (But note that table file copying methods do not work if your database contains `InnoDB` tables. `mysqlhotcopy` does not work for `InnoDB` tables because `InnoDB` does not necessarily store table contents in database directories. Also, even if the server is not actively updating data, `InnoDB` may still have modified data cached in memory and not flushed to disk.

Making Delimited-Text File Backups

To create a text file containing a table's data, you can use `SELECT * INTO OUTFILE '`*`file_name`*`' FROM `*`tbl_name`*. The file is created on the MySQL server host, not the client host. For this statement, the output file cannot already exist because permitting files to be overwritten constitutes a security risk. See Section 12.2.7, "`SELECT` Syntax". This method works for any kind of data file, but saves only table data, not the table structure.

Another way to create text data files (along with files containing `CREATE TABLE` statements for the backed up tables) is to use `mysqldump` with the `--tab` [299] option. See Section 6.4.3, "Dumping Data in Delimited-Text Format with `mysqldump`".

To reload a delimited-text data file, use `LOAD DATA INFILE` or `mysqlimport`.

Making Backups with `mysqldump` or `mysqlhotcopy`

The `mysqldump` program and the `mysqlhotcopy` script can make backups. `mysqldump` is more general because it can back up all kinds of tables. `mysqlhotcopy` works only with some storage engines. (See Section 6.4, "Using `mysqldump` for Backups", and Section 4.6.8, "`mysqlhotcopy` — A Database Backup Program".)

For `InnoDB` tables, it is possible to perform an online backup that takes no locks on tables using the `--single-transaction` [298] option to `mysqldump`. See Section 6.3.1, "Establishing a Backup Policy".

Making Incremental Backups by Enabling the Binary Log

MySQL supports incremental backups: You must start the server with the `--log-bin` [1181] option to enable binary logging; see Section 5.3.4, "The Binary Log". The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using `FLUSH LOGS`. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in Section 6.5, "Point-in-Time (Incremental) Recovery Using the Binary Log". The next time you do a full backup, you should also rotate the binary log using `FLUSH LOGS`, `mysqldump --flush-logs`, or `mysqlhotcopy --flushlog`. See Section 4.5.4, "`mysqldump` — A Database Backup Program", and Section 4.6.8, "`mysqlhotcopy` — A Database Backup Program".

Making Backups Using Replication Slaves

If you have performance problems with your master server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See Using Replication for Backups.

If you are backing up a slave replication server, you should back up its `master.info` and `relay-log.info` files when you back up the slave's databases, regardless of the backup method you choose. These information files are always needed to resume replication after you restore the slave's data. If your slave is replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The location of this directory is the value of the `--slave-load-tmpdir` [1179] option. If the server was not started with that option, the directory location is the value of the `tmpdir` [432] system variable.

Recovering Corrupt Tables

If you have to restore `MyISAM` tables that have become corrupt, try to recover them using `REPAIR TABLE` or `myisamchk -r` first. That should work in 99.9% of all cases. If `myisamchk` fails, see Section 6.6, "`MyISAM` Table Maintenance and Crash Recovery".

Making Backups Using a File System Snapshot

If you are using a Veritas file system, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfs snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.
5. Unmount the snapshot.

Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

6.3 Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that enables you to recover data after several types of crashes:

- Operating system crash
- Power failure
- File system crash
- Hardware problem (hard drive, motherboard, and so forth)

The following instructions assume a minimum version of MySQL 4.1.8, because some `mysqldump` options used here are not available in earlier versions.

The example commands do not include options such as `--user` [227] and `--password` [226] for the `mysqldump` and `mysql` client programs. You should include such options as necessary to enable client programs to connect to the MySQL server.

Assume that data is stored in the InnoDB storage engine, which has support for transactions and automatic crash recovery. Assume also that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The InnoDB data files might not contain consistent data due to the crash, but InnoDB reads its logs and finds in them the list of pending committed and noncommitted transactions that have not been flushed to the data files. InnoDB automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of file system crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that backups must already have been made. To make sure that is the case, design and implement a backup policy.

6.3.1 Establishing a Backup Policy

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, InnoDB Hot Backup provides online nonblocking physical backup of the InnoDB data files, and mysqldump provides online logical backup. This discussion uses mysqldump.

Assume that we make a full backup of all our InnoDB tables in all databases using the following command on Sunday at 1 p.m., when load is low:

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

The resulting .sql file produced by mysqldump contains a set of SQL INSERT statements that can be used to reload the dumped tables at a later time.

This backup operation acquires a global read lock on all tables at the beginning of the dump (using FLUSH TABLES WITH READ LOCK). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the

backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

It was assumed earlier that the tables to back up are `InnoDB` tables, so `--single-transaction` [298] uses a consistent read and guarantees that data seen by `mysqldump` does not change. (Changes made by other clients to `InnoDB` tables are not seen by the `mysqldump` process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the `MyISAM` tables in the `mysql` database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the `--log-bin` [1181] option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that was started with the `--log-bin` [1181] option and that has been running for some days, we find these MySQL binary log files:

```
-rw-rw---- 1 guilhem  guilhem   1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem  guilhem         4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem  guilhem        79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem  guilhem       508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem  guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem  guilhem    998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem  guilhem       361 Nov 14 10:07 gbichot2-bin.index
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a `mysqladmin flush-logs` command. `mysqldump` also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous `mysqldump` command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`, because the `--flush-logs` [294] option causes the server to flush its logs. The `--master-data` [296] option causes `mysqldump` to write binary log information to its output, so the resulting `.sql` dump file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Because the `mysqldump` command made a full backup, those lines mean two things:

- The dump file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or newer.
- All data changes logged after the backup are not present in the dump file, but are present in the `gbichot2-bin.000007` binary log file or newer.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a `mysqladmin flush-logs` command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another `mysqladmin flush-logs` command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the `gbichot2-bin.000008` file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Note

Deleting the MySQL binary logs with `mysqldump --delete-master-logs` can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the `PURGE BINARY LOGS` statement explains what should be verified before deleting the MySQL binary logs. See Section 12.5.1.1, “PURGE BINARY LOGS Syntax”.

6.3.2 Using Backups for Recovery

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a `--log-bin` [1181] option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the `gbichot2-bin.000009` file (and any subsequent files) at hand, and we could apply them using `mysqlbinlog` and `mysql` to restore the most recent data changes with no loss up to the moment of the crash:

```
```

```
shell> mysqlbinlog gbichot2-bin.000009 ... | mysql
```

For more information about using `mysqlbinlog` to process binary log files, see Section 6.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

6.3.3 Backup Strategy Summary

In case of an operating system crash or power failure, `InnoDB` itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin` [1181] option, or even `--log-bin=log_name` [1181], where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).
- Make periodic full backups, using the `mysqldump` command shown earlier in Section 6.3.1, “Establishing a Backup Policy”, that makes an online, nonblocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or `mysqladmin flush-logs`.

6.4 Using `mysqldump` for Backups

This section describes how to use `mysqldump` to produce dump files, and how to reload dump files. A dump file can be used in several ways:

- As a backup to enable data recovery in case of data loss.
- As a source of data for setting up replication slaves.
- As a source of data for experimentation:
  - To make a copy of a database that you can use without changing the original data.
  - To test potential upgrade incompatibilities.

`mysqldump` produces two types of output, depending on whether the `--tab` [299] option is given:

- Without `--tab` [299], `mysqldump` writes SQL statements to the standard output. This output consists of `CREATE` statements to create dumped objects (databases, tables, and so forth), and `INSERT` statements to load data into tables. The output can be saved in a file and reloaded later using `mysql` to recreate the dumped objects. Options are available to modify the format of the SQL statements.
- With `--tab` [299], `mysqldump` produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named *`tbl_name`*`.txt` in the output directory. The server also sends a `CREATE TABLE` statement for the table to `mysqldump`, which writes it as a file named *`tbl_name`*`.sql` in the output directory.

6.4.1 Dumping Data in SQL Format with `mysqldump`

This section describes how to use `mysqldump` to create SQL-format dump files. For information about reloading such dump files, see Section 6.4.2, “Reloading SQL-Format Backups”.

By default, `mysqldump` writes information as SQL statements to the standard output. You can save the output in a file:

```
```

```
shell> mysqldump [arguments] > file_name
```

To dump all databases, invoke `mysqldump` with the `--all-databases` [293] option:

```
shell> mysqldump --all-databases > dump.sql
```

To dump only specific databases, name them on the command line and use the `--databases` [294] option:

```
shell> mysqldump --databases db1 db2 db3 > dump.sql
```

The `--databases` [294] option causes all names on the command line to be treated as database names. Without this option, `mysqldump` treats the first name as a database name and those following as table names.

With `--all-databases` [293] or `--databases` [294], `mysqldump` writes `CREATE DATABASE` and `USE` statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the `--add-drop-database` [292] option as well. In this case, `mysqldump` writes a `DROP DATABASE` statement preceding each `CREATE DATABASE` statement.

To dump a single database, name it on the command line:

```
shell> mysqldump --databases test > dump.sql
```

In the single-database case, it is permissible to omit the `--databases` [294] option:

```
shell> mysqldump test > dump.sql
```

The difference between the two preceding commands is that without `--databases` [294], the dump output contains no `CREATE DATABASE` or `USE` statements. This has several implications:

- When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
- For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
- If the database to be reloaded does not exist, you must create it first.
- Because the output will contain no `CREATE DATABASE` statement, the `--add-drop-database` [292] option has no effect. If you use it, it produces no `DROP DATABASE` statement.

To dump only specific tables from a database, name them on the command line following the database name:

```
shell> mysqldump test t1 t3 t7 > dump.sql
```

6.4.2 Reloading SQL-Format Backups

To reload a dump file written by `mysqldump` that consists of SQL statements, use it as input to the `mysql` client. If the dump file was created by `mysqldump` with the `--all-databases` [293] or `--databases` [294] option, it contains `CREATE DATABASE` and `USE` statements and it is not necessary to specify a default database into which to load the data:

```
shell> mysql < dump.sql
```

Alternatively, from within mysql, use a source command:

```
mysql> source dump.sql
```

If the file is a single-database dump not containing CREATE DATABASE and USE statements, create the database first (if necessary):

```
shell> mysqladmin create db1
```

Then specify the database name when you load the dump file:

```
shell> mysql db1 < dump.sql
```

Alternatively, from within mysql, create the database, select it as the default database, and load the dump file:

```
mysql> CREATE DATABASE IF NOT EXISTS db1;
mysql> USE db1;
mysql> source dump.sql
```

6.4.3 Dumping Data in Delimited-Text Format with mysqldump

This section describes how to use mysqldump to create delimited-text dump files. For information about reloading such dump files, see Section 6.4.4, “Reloading Delimited-Text Format Backups”.

If you invoke mysqldump with the --tab=*dir_name* [299] option, it uses *dir_name* as the output directory and dumps tables individually in that directory using two files for each table. The table name is the basename for these files. For a table named t1, the files are named t1.sql and t1.txt. The .sql file contains a CREATE TABLE statement for the table. The .txt file contains the table data, one line per table row.

The following command dumps the contents of the db1 database to files in the /tmp database:

```
shell> mysqldump --tab=/tmp db1
```

The .txt files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses SELECT ... INTO OUTFILE to write the files, so you must have the FILE [491] privilege to perform this operation, and an error occurs if a given .txt file already exists.

The server sends the CREATE definitions for dumped tables to mysqldump, which writes them to .sql files. These files therefore are owned by the user who executes mysqldump.

It is best that --tab [299] be used only for dumping a local server. If you use it with a remote server, the --tab [299] directory must exist on both the local and remote hosts, and the .txt files will be written by the server in the remote directory (on the server host), whereas the .sql files will be written by mysqldump in the local directory (on the client host).

For mysqldump --tab, the server by default writes table data to .txt files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator. (These are the same defaults as for SELECT ... INTO OUTFILE.)

To enable data files to be written using a different format, `mysqldump` supports these options:

- `--fields-terminated-by=`*`str`* [294]

  The string for separating column values (default: tab).

- `--fields-enclosed-by=`*`char`* [294]

  The character within which to enclose column values (default: no character).

- `--fields-optionally-enclosed-by=`*`char`* [294]

  The character within which to enclose non-numeric column values (default: no character).

- `--fields-escaped-by=`*`char`* [294]

  The character for escaping special characters (default: no escaping).

- `--lines-terminated-by=`*`str`* [295]

  The line-termination string (default: newline).

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want `mysqldump` to quote column values within double quotation marks. To do so, specify double quote as the value for the `--fields-enclosed-by` [294] option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

```
--fields-enclosed-by='"'
```

On any platform, you can specify the value in hex:

```
--fields-enclosed-by=0x22
```

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (`\r\n`), use this command (enter it on a single line):

```
shell> mysqldump --tab=/tmp --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1
```

Should you use any of the data-formatting options to dump table data, you will need to specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

6.4.4 Reloading Delimited-Text Format Backups

For backups produced with `mysqldump --tab`, each table is represented in the output directory by an `.sql` file containing the `CREATE TABLE` statement for the table, and a `.txt` file containing the table data. To reload a table, first change location into the output directory. Then process the `.sql` file with `mysql` to create an empty table and process the `.txt` file to load the data into the table:

```
shell> mysql db1 < t1.sql
shell> mysqlimport db1 t1.txt
```

An alternative to using mysqlimport to load the data file is to use the LOAD DATA INFILE statement from within the mysql client:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

If you used any data-formatting options with mysqldump when you initially dumped the table, you must use the same options with mysqlimport or LOAD DATA INFILE to ensure proper interpretation of the data file contents:

```
shell> mysqlimport --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1 t1.txt
```

Or:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1
    -> FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r\n';
```

6.4.5 mysqldump Tips

This section surveys techniques that enable you to use mysqldump to solve specific problems:

- How to make a copy a database
- How to copy a database from one server to another
- How to dump definitions and data separately

6.4.5.1 Making a Copy of a Database

```
shell> mysqldump db1 > dump.sql
shell> mysqladmin create db2
shell> mysql db2 < dump.sql
```

Do not use --databases [294] on the mysqldump command line because that causes USE db1 to be included in the dump file, which overrides the effect of naming db2 on the mysql command line.

6.4.5.2 Copy a Database from one Server to Another

On Server 1:

```
shell> mysqldump --databases db1 > dump.sql
```

Copy the dump file from Server 1 to Server 2.

On Server 2:

```
shell> mysql < dump.sql
```

Use of --databases [294] with the mysqldump command line causes the dump file to include CREATE DATABASE and USE statements that create the database if it does exist and make it the default database for the reloaded data.

Alternatively, you can omit `--databases` [294] from the `mysqldump` command. Then you will need to create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

```
shell> mysqldump db1 > dump.sql
```

On Server 2:

```
shell> mysqladmin create db1
shell> mysql db1 < dump.sql
```

You can specify a different database name in this case, so omitting `--databases` [294] from the `mysqldump` command enables you to dump data from one database and load it into another.

6.4.5.3 Dumping Table Definitions and Content Separately

The `--no-data` [297] option tells `mysqldump` not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the `--no-create-info` [297] option tells `mysqldump` to suppress `CREATE` statements from the output, so that the dump file contains only table data.

For example, to dump table definitions and data separately for the `test` database, use these commands:

```
shell> mysqldump --no-data test > dump-defs.sql
shell> mysqldump --no-create-info test > dump-data.sql
```

6.4.5.4 Using `mysqldump` to Test for Upgrade Incompatibilities

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

```
shell> mysqldump --all-databases --no-data > dump-defs.sql
```

On the upgraded server:

```
shell> mysql < dump-defs.sql
```

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

```
shell> mysqldump --all-databases --no-create-info > dump-data.sql
```

On the upgraded server:

```
shell> mysql < dump-data.sql
```

Now check the table contents and run some test queries.

6.5 Point-in-Time (Incremental) Recovery Using the Binary Log

Point-in-time recovery refers to recovery of data changes made since a given point in time. Typically, this type of recovery is performed after restoring a full backup that brings the server to its state as of the time the backup was made. (The full backup can be made in several ways, such as those listed in Section 6.2, “Database Backup Methods”.) Point-in-time recovery then brings the server up to date incrementally from the time of the full backup to a more recent time.

Point-in-time recovery is based on these principles:

- The source of information for point-in-time recovery is the set of incremental backups represented by the binary log files generated subsequent to the full backup operation. Therefore, the server must be started with the `--log-bin` [1181] option to enable binary logging (see Section 5.3.4, “The Binary Log”).

  To restore data from the binary log, you must know the name and location of the current binary log files. By default, the server creates binary log files in the data directory, but a path name can be specified with the `--log-bin` [1181] option to place the files in a different location. Section 5.3.4, “The Binary Log”.

  To see a listing of all binary log files, use this statement:

  ```
  mysql> SHOW BINARY LOGS;
  ```

  To determine the name of the current binary log file, issue the following statement:

  ```
  mysql> SHOW MASTER STATUS;
  ```

- The `mysqlbinlog` utility converts the events in the binary log files from binary format to text so that they can be executed or viewed. `mysqlbinlog` has options for selecting sections of the binary log based on event times or position of events within the log. See Section 4.6.6, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

- Executing events from the binary log causes the data modifications they represent to be redone. This enables recovery of data changes for a given span of time. To execute events from the binary log, process `mysqlbinlog` output using the `mysql` client:

  ```
  shell> mysqlbinlog binlog_files | mysql -u root -p
  ```

- Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send `mysqlbinlog` output into a paging program:

  ```
  shell> mysqlbinlog binlog_files | more
  ```

  Alternatively, save the output in a file and view the file in a text editor:

  ```
  shell> mysqlbinlog binlog_files > tmpfile
  shell> ... edit tmpfile ...
  ```

- Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental `DROP DATABASE`. You can delete from the file any statements not to be executed before executing its contents. After editing the file, execute the contents as follows:

```
shell> mysql -u root -p < tmpfile
```

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

6.5.1 Point-in-Time Recovery Using Event Times

To indicate the start and end times for recovery, specify the `--start-datetime` [341] and `--stop-datetime` [342] options for `mysqlbinlog`, in `DATETIME` format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
shell> mysqlbinlog --stop-datetime="2005-04-20 9:59:59" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-datetime` [342] option. If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run `mysqlbinlog` again with a start date and time, like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 10:01:00" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two `mysqlbinlog` commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on.

To use this method of point-in-time recovery, you should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
```

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the `/tmp/mysql_restore.sql` file with a text editor to examine it.

Excluding specific changes by specifying times for `mysqlbinlog` does not work well if multiple statements executed at the same time as the one to be excluded.

6.5.2 Point-in-Time Recovery Using Event Positions

Instead of specifying dates and times, the `--start-position` [342] and `--stop-position` [342] options for `mysqlbinlog` can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around the same time as a damaging SQL statement. To determine the position numbers, run `mysqlbinlog` for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 9:55:00" \
        --stop-datetime="2005-04-20 10:05:00" \
        /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the `/tmp` directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you do not want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as `log_pos` followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \
        | mysql -u root -p

shell> mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \
        | mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of `mysqlbinlog` includes `SET TIMESTAMP` statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

6.6 `MyISAM` Table Maintenance and Crash Recovery

This section discusses how to use `myisamchk` to check or repair `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes). The same concepts apply to using `isamchk` to check or repair `ISAM` tables (tables that have `.ISD` and `.ISM` files for storing data and indexes). For general `myisamchk` or `isamchk` background, see Section 4.6.2, “`myisamchk` — MyISAM Table-Maintenance Utility”. Other table-repair information can be found at Section 2.11.4, “Rebuilding or Repairing Tables or Indexes”.

You can use `myisamchk` to check, repair, or optimize database tables. The following sections describe how to perform these operations and how to set up a table maintenance schedule. For information about using `myisamchk` to get information about your tables, see Section 4.6.2.5, “Obtaining Table Information with `myisamchk`”.

Even though table repair with `myisamchk` is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

myisamchk operations that affect indexes can cause FULLTEXT indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in Section 4.6.2.1, “myisamchk General Options”.

MyISAM table maintenance can also be done using the SQL statements that perform operations similar to what myisamchk can do:

- To check MyISAM tables, use CHECK TABLE.
- To repair MyISAM tables, use REPAIR TABLE.
- To optimize MyISAM tables, use OPTIMIZE TABLE.
- To analyze MyISAM tables, use ANALYZE TABLE.

For additional information about these statements, see Section 12.4.2, “Table Maintenance Statements”.

These statements were introduced in different versions, but all are available from MySQL 3.23.14 on. These statements can be used directly or by means of the mysqlcheck client program. One advantage of these statements over myisamchk is that the server does all the work. With myisamchk, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between myisamchk and the server.

6.6.1 Using myisamchk for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.

For an explanation of how MyISAM tables can become corrupted, see Section 13.1.4, “MyISAM Table Problems”.

If you run mysqld with external locking disabled (which is the default as of MySQL 4.0), you cannot reliably use myisamchk to check a table when mysqld is using the same table. If you can be certain that no one will access the tables through mysqld while you run myisamchk, you only have to execute mysqladmin flush-tables before you start checking the tables. If you cannot guarantee this, you must stop mysqld while you check the tables. If you run myisamchk to check tables that mysqld is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use myisamchk to check tables at any time. In this case, if the server tries to update a table that myisamchk is using, the server will wait for myisamchk to finish before it continues.

If you use myisamchk to repair or optimize tables, you *must* always ensure that the mysqld server is not using the table (this also applies if external locking is disabled). If you do not stop mysqld, you should at least do a mysqladmin flush-tables before you run myisamchk. Your tables *may become corrupted* if the server and myisamchk access the tables simultaneously.

When performing crash recovery, it is important to understand that each MyISAM table *tbl_name* in a database corresponds to the three files in the database directory shown in the following table.

File	Purpose
tbl_name.frm	Definition (format) file
tbl_name.MYD	Data file
tbl_name.MYI	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

myisamchk works by creating a copy of the .MYD data file row by row. It ends the repair stage by removing the old .MYD file and renaming the new file to the original file name. If you use --quick [317], myisamchk does not create a temporary .MYD file, but instead assumes that the .MYD file is correct and generates only a new index file without touching the .MYD file. This is safe, because myisamchk automatically detects whether the .MYD file is corrupt and aborts the repair if it is. You can also specify the --quick [317] option twice to myisamchk. In this case, myisamchk does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the .MYD file. Normally the use of two --quick [317] options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running myisamchk.

6.6.2 How to Check MyISAM Tables for Errors

To check a MyISAM table, use the following commands:

- myisamchk *tbl_name*

  This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run myisamchk without options or with the -s (silent) option.

- myisamchk -m *tbl_name*

  This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- myisamchk -e *tbl_name*

  This does a complete and thorough check of all data (-e means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, myisamchk stops after the first error it finds. If you want to obtain more information, you can add the -v (verbose) option. This causes myisamchk to keep going, up through a maximum of 20 errors.

- myisamchk -e -i *tbl_name*

  This is like the previous command, but the -i option tells myisamchk to print additional statistical information.

In most cases, a simple myisamchk command with no arguments other than the table name is sufficient to check a table.

6.6.3 How to Repair MyISAM Tables

The discussion in this section describes how to use myisamchk on MyISAM tables (extensions .MYI and .MYD). If you are using ISAM tables (extensions .ISM and .ISD), you should use isamchk instead; the concepts are similar.

You can also use the CHECK TABLE and REPAIR TABLE statements to check and repair MyISAM tables. See Section 12.4.2.3, “CHECK TABLE Syntax”, and Section 12.4.2.6, “REPAIR TABLE Syntax”.

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- *tbl_name*.frm is locked against change
- Can't find file *tbl_name*.MYI (Errcode: *nnn*)
- Unexpected end of file
- Record file is crashed
- Got error *nnn* from table handler

To get more information about the error, run perror *nnn*, where *nnn* is the error number. The following example shows how to use perror to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use ALTER TABLE to increase the MAX_ROWS and AVG_ROW_LENGTH table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use SHOW CREATE TABLE.

For the other errors, you must repair your tables. myisamchk can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that mysqld runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in Section 6.6.2, “How to Check MyISAM Tables for Errors”), or you want to use the extended features that myisamchk provides.

The options that you can use for table maintenance with myisamchk and isamchk are described in Section 4.6.2, “myisamchk — MyISAM Table-Maintenance Utility”. myisamchk also has variables that you can set to control memory allocation that may improve performance. See Section 4.6.2.6, “myisamchk Memory Usage”.

If you are going to repair a table from the command line, you must first stop the mysqld server. Note that when you do mysqladmin shutdown on a remote server, the mysqld server is still available for a while after mysqladmin returns, until all statement-processing has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run myisamchk *.MYI or myisamchk -e *.MYI if you have more time. Use the -s (silent) option to suppress unnecessary information.

If the mysqld server is stopped, you should use the --update-state [316] option to tell myisamchk to mark the table as “checked.”

You have to repair only those tables for which myisamchk announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as out of memory errors), or if myisamchk crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try myisamchk -r -q *tbl_name* (-r -q means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use myisamchk -r *tbl_name* (-r means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.
3. If the preceding step fails, use myisamchk --safe-recover *tbl_name*. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

Note

If you want a repair operation to go much faster, you should set the values of the sort_buffer_size [427] and key_buffer_size [415] variables each to about 25% of your available memory when running myisamchk.

If you get unexpected errors when repairing (such as out of memory errors), or if myisamchk crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.
2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
mysql> SET autocommit=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

   If your version of MySQL does not have TRUNCATE TABLE, use DELETE FROM *tbl_name* instead.

3. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

Important

If you are using replication, you should stop it prior to performing the above procedure, since it involves file system operations, and these are not logged by MySQL.

Go back to Stage 2. `myisamchk -r -q` should work. (This should not be an endless loop.)

As of MySQL 4.0.2, you can also use the `REPAIR TABLE tbl_name USE_FRM` SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See Section 12.4.2.6, "`REPAIR TABLE` Syntax".

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file is not changed after the table is created:

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `myisamchk -r`.

2. If you do not have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, and then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

6.6.4 MyISAM Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run `myisamchk` in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE` SQL statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See Section 12.4.2.5, "`OPTIMIZE TABLE` Syntax".

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze` [318] or `-a`: Perform key distribution analysis. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use.
- `--sort-index` [319] or `-S`: Sort the index blocks. This optimizes seeks and makes table scans that use indexes faster.
- `--sort-records=index_num` [319] or `-R index_num`: Sort data rows according to a given index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index.

For a full description of all available options, see Section 4.6.2, "`myisamchk` — MyISAM Table-Maintenance Utility".

6.6.5 Setting Up a MyISAM Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair `MyISAM` tables is with the `CHECK TABLE` and `REPAIR TABLE` statements. These are available starting with MySQL 3.23.16. See Section 12.4.2, "Table Maintenance Statements".

Another way to check tables is to use `myisamchk`. For maintenance purposes, you can use `myisamchk -s`. The `-s` option (short for `--silent` [314]) causes `myisamchk` to run in silent mode, printing messages only when errors occur.

It is also a good idea to check tables when the server starts. For example, whenever the machine has done a restart in the middle of an update, you usually need to check all the tables that could have been affected. (These are “expected” crashed tables.) To cause the server to check MyISAM tables automatically, start it with the --myisam-recover [390] option, available as of MySQL 3.23.25. If your server is too old to support this option, you could add a test to mysqld_safe that runs myisamchk to check all tables that have been modified during the last 24 hours if there is an old .pid (process ID) file left after a restart. (The .pid file is created by mysqld when it starts and removed when it terminates normally. The presence of a .pid file at system startup time indicates that mysqld terminated abnormally.)

It is also a good idea to enable automatic MyISAM table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To check MyISAM tables automatically, start the server with the --myisam-recover [390] option, available as of MySQL 3.23.25. See Section 5.1.2, “Server Command Options”. If your server is too old to support this option, you could add a test to mysqld_safe that runs myisamchk to check all tables that have been modified during the last 24 hours if there is an old .pid (process ID) file left after a restart. (The .pid file is created by mysqld when it starts and removed when it terminates normally. The presence of a .pid file at system startup time indicates that mysqld terminated abnormally.)

You should also check your tables regularly during normal system operation. For example, you can run a cron job to check important tables once a week, using a line like this in a crontab file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that you can examine and repair them as necessary.

To start with, execute myisamchk -s each night on all tables that have been updated during the last 24 hours. As you see that problems occur infrequently, you can back off the checking frequency to once a week or so.

Normally, MySQL tables need little maintenance. If you are performing many updates to MyISAM tables with dynamic-sized rows (tables with VARCHAR, BLOB, or TEXT columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using OPTIMIZE TABLE on the tables in question. Alternatively, if you can stop the mysqld server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --sort_buffer_size=16M */*.MYI
```

For ISAM tables, the command is similar:

```
shell> isamchk -r -s --sort-index -O sort_buffer_size=16M */*.ISM
```

Chapter 7 Optimization

Table of Contents

This chapter explains different ways to optimize MySQL and provides examples. Optimization is a complex task because ultimately it requires understanding of the entire system to be optimized. Although it may be possible to perform some local optimizations with little knowledge of your system or application, the more optimal you want your system to become, the more you must know about it.

7.1 Optimization Overview

The most important factor in making a system fast is its basic design. You must also know what kinds of processing your system is doing, and what its bottlenecks are. In most cases, system bottlenecks arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When we have the data in main memory, we need to process it to get our result. Having small tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.
- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

7.1.1 MySQL Design Limitations and Tradeoffs

When using the `MyISAM` storage engine, MySQL uses extremely fast table locking that permits multiple readers or a single writer. The biggest problem with this storage engine occurs when you have a steady stream of mixed updates and slow selects on a single table. If this is a problem for certain tables, you can use another storage engine for them. See Chapter 13, *Storage Engines*.

MySQL can work with both transactional and nontransactional tables. To make it easier to work smoothly with nontransactional tables (which cannot roll back if something goes wrong), MySQL has the following rules. Note that these rules apply *only* when you use the `IGNORE` specifier for `INSERT` or `UPDATE`.

- All columns have default values.
- If you insert an inappropriate or out-of-range value into a column, MySQL sets the column to the “best possible value” instead of reporting an error. For numeric values, this is 0, the smallest possible value or the largest possible value. For strings, this is either the empty string or as much of the string as can be stored in the column.
- All calculated expressions return a value that can be used instead of signaling an error condition. For example, 1/0 returns `NULL`.

To change the preceding behaviors, you can enable stricter data handling by setting the server SQL mode appropriately. For more information about data handling, see Section 1.9.6, “How MySQL Deals with Constraints”, Section 5.1.6, “Server SQL Modes”, and Section 12.2.4, “`INSERT` Syntax”.

7.1.2 Designing Applications for Portability

Because all SQL servers implement different parts of standard SQL, it takes work to write portable database applications. It is very easy to achieve portability for very simple selects and inserts, but becomes more difficult the more capabilities you require. If you want an application that is fast with many database systems, it becomes even more difficult.

All database systems have some weak points. That is, they have different design compromises that lead to different behavior.

To make a complex application portable, you need to determine which SQL servers it must work with, and then determine what features those servers support. You can use the MySQL `crash-me` program to find functions, types, and limits that you can use with a selection of database servers. `crash-me` does not check for every possible feature, but it is still reasonably comprehensive, performing about 450 tests. An example of the type of information `crash-me` can provide is that you should not use column names that are longer than 18 characters if you want to be able to use Informix or DB2.

The `crash-me` program and the MySQL benchmarks are all very database independent. By taking a look at how they are written, you can get a feeling for what you must do to make your own applications database independent. The programs can be found in the `sql-bench` directory of MySQL source distributions. They are written in Perl and use the DBI database interface. Use of DBI in itself solves part of the portability problem because it provides database-independent access methods. See Section 7.1.3, “The MySQL Benchmark Suite”.

If you strive for database independence, you need to get a good feeling for each SQL server's bottlenecks. For example, MySQL is very fast in retrieving and updating rows for `MyISAM` tables, but has a problem in mixing slow readers and writers on the same table. Transactional database systems in general are not very good at generating summary tables from log tables, because in this case row locking is almost useless.

To make your application *really* database independent, you should define an easily extendable interface through which you manipulate your data. For example, C++ is available on most systems, so it makes sense to use a C++ class-based interface to the databases.

If you use some feature that is specific to a given database system (such as the `REPLACE` statement, which is specific to MySQL), you should implement the same feature for other SQL servers by coding an alternative method. Although the alternative might be slower, it enables the other servers to perform the same tasks.

With MySQL, you can use the `/*! */` syntax to add MySQL-specific keywords to a statement. The code inside `/* */` is treated as a comment (and ignored) by most other SQL servers. For information about writing comments, see Section 8.6, “Comment Syntax”.

If high performance is more important than exactness, as for some Web applications, it is possible to create an application layer that caches all results to give you even higher performance. By letting old results expire after a while, you can keep the cache reasonably fresh. This provides a method to handle high load spikes, in which case you can dynamically increase the cache size and set the expiration timeout higher until things get back to normal.

In this case, the table creation information should contain information about the initial cache size and how often the table should normally be refreshed.

An attractive alternative to implementing an application cache is to use the MySQL query cache. By enabling the query cache, the server handles the details of determining whether a query result can be reused. This simplifies your application. See Section 7.5.3, “The MySQL Query Cache”.

7.1.3 The MySQL Benchmark Suite

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly. You can get a good idea for how the benchmarks work by looking at the code and results in the `sql-bench` directory in any MySQL source distribution.

Note that this benchmark is single-threaded, so it measures the minimum time for the operations performed. We plan to add multi-threaded tests to the benchmark suite in the future.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from http://dev.mysql.com/downloads/, or use the current development source tree. (See Section 2.9.2, “Installing MySQL from a Development Source Tree”.)
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` modules installed. See Section 2.14, “Perl Installation Notes”.

After you obtain a MySQL source distribution, you can find the benchmark suite located in its `sql-bench` directory. To run the benchmark tests, build MySQL, and then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` should be the name of one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
shell> perl run-all-tests --help
```

The `crash-me` script also is located in the `sql-bench` directory. `crash-me` tries to determine what features a database system supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What data types are supported
- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a `VARCHAR` column can be

For more information about benchmark results, visit http://www.mysql.com/why-mysql/benchmarks/.

7.1.4 Using Your Own Benchmarks

You should definitely benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a “dummy” module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

For examples of portable benchmark programs, look at those in the MySQL benchmark suite. See Section 7.1.3, “The MySQL Benchmark Suite”. You can take any program from this suite and modify it

for your own needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at http://osdb.sourceforge.net/.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. Most of the time, these problems would be much easier to fix if the systems were not already in production.

To avoid problems like this, you should put some effort into benchmarking your whole application under the worst possible load. You can use Super Smack, available at http://jeremy.zawodny.com/mysql/super-smack/. As suggested by its name, it can bring a system to its knees, so make sure to use it only on your development systems.

7.2 Obtaining Query Execution Plan Information

7.2.1 Optimizing Queries with `EXPLAIN`

The `EXPLAIN` statement can be used either as a synonym for `DESCRIBE` or as a way to obtain information about how MySQL executes a `SELECT` statement:

- `EXPLAIN tbl_name` is synonymous with `DESCRIBE tbl_name` or `SHOW COLUMNS FROM tbl_name`:

```
EXPLAIN tbl_name
```

- When you precede a `SELECT` statement with the keyword `EXPLAIN`, MySQL displays information from the optimizer about the query execution plan. That is, MySQL explains how it would process the `SELECT`, including information about how tables are joined and in which order:

```
EXPLAIN [EXTENDED] SELECT select_options
```

This section describes the second use of `EXPLAIN` for obtaining query execution plan information. See also Section 12.7.2, “`EXPLAIN` Syntax”. For a description of the `DESCRIBE` and `SHOW COLUMNS` statements, see Section 12.7.1, “`DESCRIBE` Syntax”, and Section 12.4.5.5, “`SHOW COLUMNS` Syntax”.

With the help of `EXPLAIN`, you can see where you should add indexes to tables to get a faster `SELECT` that uses indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in the `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See Section 12.2.7, “`SELECT` Syntax”.)

If you have a problem with indexes not being used when you believe that they should be, you should run `ANALYZE TABLE` to update table statistics such as cardinality of keys, that can affect the choices the optimizer makes. See Section 12.4.2.1, “`ANALYZE TABLE` Syntax”.

7.2.2 `EXPLAIN` Output Format

`EXPLAIN` returns a row of information for each table used in the `SELECT` statement. The tables are listed in the output in the order that MySQL would read them while processing the query. MySQL resolves all joins using a nested-loop join method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

In MySQL version 4.1, the EXPLAIN output format was changed to work better with constructs such as UNION statements, subqueries, and derived tables. Most notable is the addition of two new columns: id and select_type. You do not see these columns when using servers older than MySQL 4.1. EXPLAIN syntax also was augmented to permit the EXTENDED keyword. When this keyword is used, EXPLAIN produces extra information that can be viewed by issuing a SHOW WARNINGS statement following the EXPLAIN statement. This information displays how the optimizer qualifies table and column names in the SELECT statement, what the SELECT looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process.

Each output row from EXPLAIN provides information about one table, and each row contains the following columns:

- id

  The SELECT identifier. This is the sequential number of the SELECT within the query.

- select_type

  The type of SELECT, which can be any of those shown in the following table.

select_type Value	Meaning
SIMPLE	Simple SELECT (not using UNION or subqueries)
PRIMARY	Outermost SELECT
UNION	Second or later SELECT statement in a UNION
DEPENDENT UNION	Second or later SELECT statement in a UNION, dependent on outer query
UNION RESULT	Result of a UNION.
SUBQUERY	First SELECT in subquery
DEPENDENT SUBQUERY	First SELECT in subquery, dependent on outer query
DERIVED	Derived table SELECT (subquery in FROM clause)
UNCACHEABLE SUBQUERY	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query

  DEPENDENT typically signifies the use of a correlated subquery. See Section 12.2.8.7, “Correlated Subqueries”.

  DEPENDENT SUBQUERY evaluation differs from UNCACHEABLE SUBQUERY evaluation. For DEPENDENT SUBQUERY, the subquery is re-evaluated only once for each set of different values of the variables from its outer context. For UNCACHEABLE SUBQUERY, the subquery is re-evaluated for each row of the outer context. Cacheability of subqueries is subject to the restrictions detailed in Section 7.5.3.1, “How the Query Cache Operates”. For example, referring to user variables makes a subquery uncacheable.

- table

  The table to which the row of output refers.

- type

  The join type. The different join types are listed here, ordered from the best type to the worst:

  - system [566]

    The table has only one row (= system table). This is a special case of the const [567] join type.

- `const` [567]

  The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` [567] tables are very fast because they are read only once.

  `const` [567] is used when you compare all parts of a `PRIMARY KEY` or `UNIQUE` index to constant values. In the following queries, *`tbl_name`* can be used as a `const` [567] table:

  ```
  SELECT * FROM tbl_name WHERE primary_key=1;

  SELECT * FROM tbl_name
    WHERE primary_key_part1=1 AND primary_key_part2=2;
  ```

- `eq_ref` [567]

  One row is read from this table for each combination of rows from the previous tables. Other than the `system` [566] and `const` [567] types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a `PRIMARY KEY` or `UNIQUE NOT NULL` index.

  `eq_ref` [567] can be used for indexed columns that are compared using the `=` operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an `eq_ref` [567] join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column=other_table.column;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column_part1=other_table.column
    AND ref_table.key_column_part2=1;
  ```

- `ref` [567]

  All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref` [567] is used if the join uses only a leftmost prefix of the key or if the key is not a `PRIMARY KEY` or `UNIQUE` index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

  `ref` [567] can be used for indexed columns that are compared using the `=` or `<=>` operator. In the following examples, MySQL can use a `ref` [567] join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table WHERE key_column=expr;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column=other_table.column;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column_part1=other_table.column
    AND ref_table.key_column_part2=1;
  ```

- `fulltext` [567]

  The join is performed using a `FULLTEXT` index.

- `ref_or_null` [567]

This join type is like ref [567], but with the addition that MySQL does an extra search for rows that contain NULL values. This join type optimization was added for MySQL 4.1.1 and is used mostly when resolving subqueries. In the following examples, MySQL can use a ref_or_null [567] join to process *ref_table*:

```
SELECT * FROM ref_table
  WHERE key_column=expr OR key_column IS NULL;
```

See Section 7.3.1.4, “IS NULL Optimization”.

- unique_subquery [568]

  This type replaces ref [567] for some IN subqueries of the following form:

  ```
  value IN (SELECT primary_key FROM single_table WHERE some_expr)
  ```

  unique_subquery [568] is just an index lookup function that replaces the subquery completely for better efficiency.

- index_subquery [568]

  This join type is similar to unique_subquery [568]. It replaces IN subqueries, but it works for nonunique indexes in subqueries of the following form:

  ```
  value IN (SELECT key_column FROM single_table WHERE some_expr)
  ```

- range [568]

  Only rows that are in a given range are retrieved, using an index to select the rows. The key column in the output row indicates which index is used. The key_len contains the longest key part that was used. The ref column is NULL for this type.

  range [568] can be used when a key column is compared to a constant using any of the = [782], <> [782], > [783], >= [782], < [782], <= [782], IS NULL [783], <=> [782], BETWEEN [783], or IN() [784] operators:

  ```
  SELECT * FROM tbl_name
    WHERE key_column = 10;

  SELECT * FROM tbl_name
    WHERE key_column BETWEEN 10 and 20;

  SELECT * FROM tbl_name
    WHERE key_column IN (10,20,30);

  SELECT * FROM tbl_name
    WHERE key_part1= 10 AND key_part2 IN (10,20,30);
  ```

- index [568]

  This join type is the same as ALL, except that only the index tree is scanned. This usually is faster than ALL because the index file usually is smaller than the data file.

  MySQL can use this join type when the query uses only columns that are part of a single index.

- ALL

  A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked const [567], and usually *very* bad in all other cases. Normally, you can avoid ALL by adding indexes that enable row retrieval from the table based on constant values or column values from earlier tables.

- possible_keys

  The possible_keys column indicates which indexes MySQL can choose from use to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from EXPLAIN. That means that some of the keys in possible_keys might not be usable in practice with the generated table order.

  If this column is NULL, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the WHERE clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with EXPLAIN again. See Section 12.1.2, “ALTER TABLE Syntax”.

  To see what indexes a table has, use SHOW INDEX FROM *tbl_name*.

- key

  The key column indicates the key (index) that MySQL actually decided to use. If MySQL decides to use one of the possible_keys indexes to look up rows, that index is listed as the key value.

  It is possible that key will name an index that is not present in the possible_keys value. This can happen if none of the possible_keys indexes are suitable for looking up rows, but all the columns selected by the query are columns of some other index. That is, the named index covers the selected columns, so although it is not used to determine which rows to retrieve, an index scan is more efficient than a data row scan.

  For InnoDB, a secondary index might cover the selected columns even if the query also selects the primary key because InnoDB stores the primary key value with each secondary index. If key is NULL, MySQL found no index to use for executing the query more efficiently.

  To force MySQL to use or ignore an index listed in the possible_keys column, use FORCE INDEX, USE INDEX, or IGNORE INDEX in your query. See Section 12.2.7.2, “Index Hint Syntax”.

  For MyISAM and BDB tables, running ANALYZE TABLE helps the optimizer choose better indexes. For MyISAM tables, myisamchk --analyze does the same. See Section 12.4.2.1, “ANALYZE TABLE Syntax”, and Section 6.6, “MyISAM Table Maintenance and Crash Recovery”.

- key_len

  The key_len column indicates the length of the key that MySQL decided to use. The length is NULL if the key column says NULL. Note that the value of key_len enables you to determine how many parts of a multiple-part key MySQL actually uses.

- ref

  The ref column shows which columns or constants are compared to the index named in the key column to select rows from the table.

- rows

The rows column indicates the number of rows MySQL believes it must examine to execute the query.

For InnoDB tables, this number is an estimate, and may not always be exact.

- Extra

  This column contains additional information about how MySQL resolves the query. The following list explains the values that can appear in this column. If you want to make your queries as fast as possible, you should look out for Extra values of Using filesort and Using temporary.

  - const row not found

    For a query such as SELECT ... FROM *tbl_name*, the table was empty.

  - Distinct

    MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

  - Impossible HAVING

    The HAVING clause is always false and cannot select any rows.

  - Impossible WHERE

    The WHERE clause is always false and cannot select any rows.

  - Impossible WHERE noticed after reading const tables

    MySQL has read all const [567] (and system [566]) tables and notice that the WHERE clause is always false.

  - No matching min/max row

    No row satisfies the condition for a query such as SELECT MIN(...) FROM ... WHERE *condition*.

  - no matching row in const table

    For a query with a join, there was an empty table or a table with no rows satisfying a unique index condition.

  - No tables used

    The query has no FROM clause, or has a FROM DUAL clause.

  - Not exists

    MySQL was able to do a LEFT JOIN optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the LEFT JOIN criteria. Here is an example of the type of query that can be optimized this way:

    ```
    SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
      WHERE t2.id IS NULL;
    ```

Assume that t2.id is defined as NOT NULL. In this case, MySQL scans t1 and looks up the rows in t2 using the values of t1.id. If MySQL finds a matching row in t2, it knows that t2.id can never be NULL, and does not scan through the rest of the rows in t2 that have the same id value. In other words, for each row in t1, MySQL needs to do only a single lookup in t2, regardless of how many rows actually match in t2.

- Range checked for each record (index map: *N*)

  MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a range [568] access method to retrieve rows. The applicability criteria are as described in Section 7.3.1.3, "Range Optimization", with the exception that all column values for the preceding table are known and considered to be constants.

  This is not very fast, but is faster than performing a join with no index at all.

  Indexes are numbered beginning with 1, in the same order as shown by SHOW INDEX for the table. The index map value *N* is a bitmask value that indicates which indexes are candidates. For example, a value of 0x19 (binary 11001) means that indexes 1, 4, and 5 will be considered.

- Select tables optimized away

  The query contained only aggregate functions (MIN() [884], MAX() [884]) that were all resolved using an index, or COUNT(*) [882] for MyISAM, and no GROUP BY clause. The optimizer determined that only one row should be returned.

- unique row not found

  For a query such as SELECT ... FROM *tbl_name*, no rows satisfy the condition for a UNIQUE index or PRIMARY KEY on the table.

- Using filesort

  MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the WHERE clause. The keys then are sorted and the rows are retrieved in sorted order. See Section 7.3.1.7, "ORDER BY Optimization".

- Using index

  The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

- Using index for group-by

  Similar to the Using index table access method, Using index for group-by indicates that MySQL found an index that can be used to retrieve all columns of a GROUP BY or DISTINCT query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see Section 7.3.1.8, "GROUP BY Optimization".

- Using temporary

To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains `GROUP BY` and `ORDER BY` clauses that list columns differently.

- `Using where`

  A `WHERE` clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the `Extra` value is not `Using where` and the table join type is `ALL` or `index` [568]. Even if you are using an index for all parts of a `WHERE` clause, you may see `Using where` if the column can be `NULL`.

You can get a good indication of how good a join is by taking the product of the values in the `rows` column of the `EXPLAIN` output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the `max_join_size` [419] system variable, this row product also is used to determine which multiple-table `SELECT` statements to execute and which to abort. See Section 7.8.2, “Tuning Server Parameters”.

The following example shows how a multiple-table join can be optimized progressively based on the information provided by `EXPLAIN`.

Suppose that you have the `SELECT` statement shown here and that you plan to examine it using `EXPLAIN`:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
               tt.ProjectReference, tt.EstimatedShipDate,
               tt.ActualShipDate, tt.ClientID,
               tt.ServiceCodes, tt.RepetitiveID,
               tt.CurrentProcess, tt.CurrentDPPerson,
               tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
               et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
          AND tt.ActualPC = et.EMPLOYID
          AND tt.AssignedPC = et_1.EMPLOYID
          AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows.

Table	Column	Data Type
`tt`	`ActualPC`	`CHAR(10)`
`tt`	`AssignedPC`	`CHAR(10)`
`tt`	`ClientID`	`CHAR(10)`
`et`	`EMPLOYID`	`CHAR(15)`
`do`	`CUSTNMBR`	`CHAR(15)`

- The tables have the following indexes.

Table	Index
`tt`	`ActualPC`
`tt`	`AssignedPC`
`tt`	`ClientID`
`et`	`EMPLOYID` (primary key)

Table	Index
do	CUSTNMBR (primary key)

- The tt.ActualPC values are not evenly distributed.

Initially, before any optimizations have been performed, the EXPLAIN statement produces the following information:

```
table type possible_keys key  key_len ref  rows  Extra
et    ALL  PRIMARY       NULL NULL    NULL 74
do    ALL  PRIMARY       NULL NULL    NULL 2135
et_1  ALL  PRIMARY       NULL NULL    NULL 74
tt    ALL  AssignedPC,   NULL NULL    NULL 3872
           ClientID,
           ActualPC
      Range checked for each record (index map: 0x23)
```

Because type is ALL for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is 74 × 2135 × 74 × 3872 = 45,268,558,720 rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. (For ISAM tables, indexes may not be used at all unless the columns are declared the same.) In this context, VARCHAR and CHAR are considered the same if they are declared as the same size. tt.ActualPC is declared as CHAR(10) and et.EMPLOYID is CHAR(15), so there is a length mismatch.

To fix this disparity between column lengths, use ALTER TABLE to lengthen ActualPC from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now tt.ActualPC and et.EMPLOYID are both VARCHAR(15). Executing the EXPLAIN statement again produces this result:

```
table type   possible_keys key     key_len ref         rows    Extra
tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using
             ClientID,                                         where
             ActualPC
do    ALL    PRIMARY       NULL    NULL    NULL        2135
      Range checked for each record (index map: 0x1)
et_1  ALL    PRIMARY       NULL    NULL    NULL        74
      Range checked for each record (index map: 0x1)
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC 1
```

This is not perfect, but is much better: The product of the rows values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the tt.AssignedPC = et_1.EMPLOYID and tt.ClientID = do.CUSTNMBR comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
    ->                MODIFY ClientID   VARCHAR(15);
```

After that modification, EXPLAIN produces the output shown here:

```
table type   possible_keys key      key_len ref           rows Extra
et    ALL    PRIMARY       NULL     NULL    NULL          74
tt    ref    AssignedPC,   ActualPC 15      et.EMPLOYID   52   Using
             ClientID,                                        where
             ActualPC
et_1  eq_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY  15      tt.ClientID   1
```

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and `EXPLAIN` produces this result:

```
table type   possible_keys key     key_len ref           rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using
             ClientID,                                        where
             ActualPC
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC   1
et_1  eq_ref PRIMARY       PRIMARY 15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY 15      tt.ClientID   1
```

Note that the `rows` column in the output from `EXPLAIN` is an educated guess from the MySQL join optimizer. You should check whether the numbers are even close to the truth by comparing the `rows` product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause.

It is possible in some cases to execute statements that modify data when `EXPLAIN SELECT` is used with a subquery; for more information, see Section 12.2.8.8, “Subqueries in the `FROM` Clause”.

7.2.3 Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: `log(row_count) / log(index_block_length / 3 * 2 / (index_length + data_pointer_length)) + 1`.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with a key value length of three bytes (the size of `MEDIUMINT`), the formula indicates `log(500,000)/log(1024/3*2/(3+4)) + 1` = `4` seeks.

This index would require storage of about 500,000 * 7 * 3/2 = 5.2MB (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

Note that the preceding discussion does not mean that your application performance slowly degenerates by log N. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are only bound by disk-seeks (which increase by log *N*). To avoid this, increase

the key cache size as the data grows. For MyISAM tables, the key cache size is controlled by the key_buffer_size [415] system variable. See Section 7.8.2, “Tuning Server Parameters”.

7.3 Optimizing SQL Statements

7.3.1 Optimizing SELECT Statements

First, one factor affects all statements: The more complex your permissions setup, the more overhead you have. Using simpler permissions when you issue GRANT statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the tables_priv and columns_priv tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, it may be worth the time to use a simplified grant structure to reduce permission-checking overhead.

If your problem is with a specific MySQL expression or function, you can perform a timing test by invoking the BENCHMARK() [870] function using the mysql client program. Its syntax is BENCHMARK(*loop_count*,*expression*) [870]. The return value is always zero, but mysql prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

All MySQL functions should be highly optimized, but there may be some exceptions. BENCHMARK() [870] is an excellent tool for finding out if some function is a problem for your queries.

7.3.1.1 Speed of SELECT Statements

In general, when you want to make a slow SELECT ... WHERE query faster, the first thing to check is whether you can add an index. All references between different tables should usually be done with indexes. You can use the EXPLAIN statement to determine which indexes are used for a SELECT. See Section 7.2.1, “Optimizing Queries with EXPLAIN”, and Section 7.4.3, “How MySQL Uses Indexes”.

Some general tips for speeding up queries on MyISAM tables:

- To help MySQL better optimize queries, use ANALYZE TABLE or run myisamchk --analyze on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a nonconstant expression. You can check the result from the table analysis by using SHOW INDEX FROM *tbl_name* and examining the Cardinality value. myisamchk --description --verbose shows index distribution information.
- To sort an index and data according to an index, use myisamchk --sort-index --sort-records=1 (assuming that you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.

7.3.1.2 WHERE Clause Optimization

This section discusses optimizations that can be made for processing WHERE clauses. The examples use SELECT statements, but the same optimizations apply for WHERE clauses in DELETE and UPDATE statements.

Work on the MySQL optimizer is ongoing, so this section is incomplete. MySQL performs a great many optimizations, not all of which are documented here.

Some of the optimizations performed by MySQL follow:

- Removal of unnecessary parentheses:

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
   (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- COUNT(*) [882] on a single table without a WHERE is retrieved directly from the table information for MyISAM and MEMORY (HASH) tables. This is also done for any NOT NULL expression when used with only one table.
- Early detection of invalid constant expressions. MySQL quickly detects that some SELECT statements are impossible and returns no rows.
- HAVING is merged with WHERE if you do not use GROUP BY or aggregate functions (COUNT() [882], MIN() [884], and so on).
- For each table in a join, a simpler WHERE is constructed to get a fast WHERE evaluation for the table and also to skip rows as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
  - An empty table or a table with one row.
  - A table that is used with a WHERE clause on a PRIMARY KEY or a UNIQUE index, where all index parts are compared to constant expressions and are defined as NOT NULL.

  All of the following tables are used as constant tables:

```
SELECT * FROM t WHERE primary_key=1;
SELECT * FROM t1,t2
  WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in ORDER BY and GROUP BY clauses come from the same table, that table is preferred first when joining.

- If there is an ORDER BY clause and a different GROUP BY clause, or if the ORDER BY or GROUP BY contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the SQL_SMALL_RESULT option, MySQL uses an in-memory temporary table.
- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.
- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are numeric, only the index tree is used to resolve the query.
- Before each row is output, those that do not match the HAVING clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;

SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

7.3.1.3 Range Optimization

The range [568] access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections give a detailed description of how intervals are extracted from the WHERE clause.

The Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the WHERE clause, so we speak of *range conditions* rather than “intervals.”

The definition of a range condition for a single-part index is as follows:

- For both BTREE and HASH indexes, comparison of a key part with a constant value is a range condition when using the = [782], <=> [782], IN() [784], IS NULL [783], or IS NOT NULL [783] operators.
- Additionally, for BTREE indexes, comparison of a key part with a constant value is a range condition when using the > [783], < [782], >= [782], <= [782], BETWEEN [783], != [782], or <> [782] operators, or LIKE [804] comparisons if the argument to LIKE [804] is a constant string that does not start with a wildcard character.
- For all types of indexes, multiple range conditions combined with OR [787] or AND [787] form a range condition.

“Constant value” in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a const [567] or system [566] table from the same join
- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the WHERE clause:

```
SELECT * FROM t1
  WHERE key_col > 1
  AND key_col < 10;

SELECT * FROM t1
  WHERE key_col = 1
  OR key_col IN (15,18,20);

SELECT * FROM t1
  WHERE key_col LIKE 'ab%'
  OR key_col BETWEEN 'bar' AND 'foo';
```

Note that some nonconstant values may be converted to constants during the constant propagation phase.

MySQL tries to extract range conditions from the WHERE clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where key1 is an indexed column and nonkey is not indexed:

```
SELECT * FROM t1 WHERE
  (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
  (key1 < 'bar' AND nonkey = 4) OR
  (key1 < 'uux' AND key1 > 'z');
```

The extraction process for key key1 is as follows:

1. Start with original WHERE clause:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
   (key1 < 'bar' AND nonkey = 4) OR
   ```

```
(key1 < 'uux' AND key1 > 'z')
```

2. Remove `nonkey = 4` and `key1 LIKE '%b'` because they cannot be used for a range scan. The correct way to remove them is to replace them with `TRUE`, so that we do not miss any matching rows when doing the range scan. Having replaced them with `TRUE`, we get:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
(key1 < 'bar' AND TRUE) OR
(key1 < 'uux' AND key1 > 'z')
```

3. Collapse conditions that are always true or false:

   - `(key1 LIKE 'abcde%' OR TRUE)` is always true
   - `(key1 < 'uux' AND key1 > 'z')` is always false

   Replacing these conditions with constants, we get:

```
(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
```

   Removing unnecessary `TRUE` and `FALSE` constants, we obtain:

```
(key1 < 'abc') OR (key1 < 'bar')
```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

```
(key1 < 'bar')
```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the `WHERE` clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full `WHERE` clause.

The range condition extraction algorithm can handle nested `AND` [787]/`OR` [787] constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in `WHERE` clause.

The Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as `key1(key_part1, key_part2, key_part3)`, and the following set of key tuples listed in key order:

```
key_part1  key_part2  key_part3
  NULL       1          'abc'
  NULL       1          'xyz'
  NULL       2          'foo'
   1         1          'abc'
   1         1          'xyz'
   1         2          'abc'
   2         1          'aaa'
```

The condition `key_part1 = 1` defines this interval:

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition `key_part3 = 'abc'` does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For HASH indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

  ```
      key_part1 cmp const1
  AND key_part2 cmp const2
  AND ...
  AND key_partN cmp constN;
  ```

  Here, `const1`, `const2`, ... are constants, `cmp` is one of the = [782], <=> [782], or IS NULL [783] comparison operators, and the conditions cover all index parts. (That is, there are `N` conditions, one for each part of an `N`-part index.) For example, the following is a range condition for a three-part HASH index:

  ```
  key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
  ```

  For the definition of what is considered to be a constant, see The Range Access Method for Single-Part Indexes.

- For a BTREE index, an interval might be usable for conditions combined with AND [787], where each condition compares a key part with a constant value using = [782], <=> [782], IS NULL [783], > [783], < [782], >= [782], <= [782], != [782], <> [782], BETWEEN [783], or LIKE `'pattern'` [804] (where `'pattern'` does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if <> [782] or != [782] is used). For example, for this condition:

  ```
  key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
  ```

  The single interval is:

  ```
  ('foo',10,10) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
  ```

  It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value `('foo', 11, 0)`, which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with OR [787], they form a condition that covers a set of rows contained within the union of their intervals. If the conditions are combined with AND [787], they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

  ```
  (key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
  ```

  The intervals are:

  ```
  (1,-inf) < (key_part1,key_part2) < (1,2)
  (5,-inf) < (key_part1,key_part2)
  ```

In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The key_len column in the EXPLAIN output indicates the maximum length of the key prefix used.

In some cases, key_len may indicate that a key part was used, but that might be not what you would expect. Suppose that key_part1 and key_part2 can be NULL. Then the key_len column displays two key part lengths for the following condition:

```
key_part1 >= 1 AND key_part2 < 2
```

But, in fact, the condition is converted to this:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

The Range Access Method for Single-Part Indexes, describes how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index. Analogous steps are performed for range conditions on multiple-part indexes.

7.3.1.4 IS NULL Optimization

MySQL can perform the same optimization on col_name IS NULL [783] that it can use for col_name = constant_value. For example, MySQL can use indexes and ranges to search for NULL with IS NULL [783].

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
  WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a WHERE clause includes a col_name IS NULL [783] condition for a column that is declared as NOT NULL, that expression is optimized away. This optimization does not occur in cases when the column might produce NULL anyway; for example, if it comes from a table on the right side of a LEFT JOIN.

MySQL 4.1.1 and up can also optimize the combination col_name = expr OR col_name IS NULL, a form that is common in resolved subqueries. EXPLAIN shows ref_or_null [567] when this optimization is used.

This optimization can handle one IS NULL [783] for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns a and b of table t2:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1, t2
  WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
  WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);
```

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
  OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

ref_or_null [567] works by first doing a read on the reference key, and then a separate search for rows with a NULL key value.

Note that the optimization can handle only one IS NULL [783] level. In the following query, MySQL uses key lookups only on the expression (t1.a=t2.a AND t2.a IS NULL) and is not able to use the key part on b:

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL)
  OR (t1.b=t2.b AND t2.b IS NULL);
```

7.3.1.5 LEFT JOIN and RIGHT JOIN Optimization

MySQL implements a *A* LEFT JOIN *B* join_condition as follows:

- Table *B* is set to depend on table *A* and all tables on which *A* depends.
- Table *A* is set to depend on all tables (except *B*) that are used in the LEFT JOIN condition.
- The LEFT JOIN condition is used to decide how to retrieve rows from table *B*. (In other words, any condition in the WHERE clause is not used.)
- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependence, MySQL issues an error.
- All standard WHERE optimizations are performed.
- If there is a row in *A* that matches the WHERE clause, but there is no row in *B* that matches the ON condition, an extra *B* row is generated with all columns set to NULL.
- If you use LEFT JOIN to find rows that do not exist in some table and you have the following test: *col_name* IS NULL in the WHERE part, where *col_name* is a column that is declared as NOT NULL, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the LEFT JOIN condition.

The implementation of RIGHT JOIN is analogous to that of LEFT JOIN with the roles of the tables reversed.

The join optimizer calculates the order in which tables should be joined. The table read order forced by LEFT JOIN or STRAIGHT_JOIN helps the join optimizer do its work much more quickly, because there are fewer table permutations to check. Note that this means that if you do a query of the following type, MySQL does a full scan on b because the LEFT JOIN forces it to be read before d:

```
SELECT *
  FROM a JOIN b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)
  WHERE b.key=d.key;
```

The fix in this case is reverse the order in which a and b are listed in the FROM clause:

```
SELECT *
  FROM b JOIN a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)
```

```
  WHERE b.key=d.key;
```

Starting from 4.0.14, for a LEFT JOIN , if the WHERE condition is always false for the generated NULL row, the LEFT JOIN is changed to a normal join. For example, the WHERE clause would be false in the following query if t2.column1 were NULL:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to a normal join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

This can be made faster because MySQL can use table t2 before table t1 if doing so would result in a better query plan. To provide a hint about the table join order, use STRAIGHT_JOIN. (See Section 12.2.7, “SELECT Syntax”.)

7.3.1.6 Nested-Loop Join Algorithms

MySQL executes joins between tables using a nested-loop algorithm or variations on it.

Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables t1, t2, and t3 is to be executed using the following join types:

```
Table   Join Type
t1      range
t2      ref
t3      ALL
```

If a simple NLJ algorithm is used, the join would be processed like this:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions,
      send to client
    }
  }
}
```

Because the NLJ algorithm passes rows one at a time from outer loops to inner loops, tables processed in the inner loops typically are read many times.

Block Nested-Loop Join Algorithm

A Block Nested-Loop (BNL) Join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all 10 rows in the buffer. The reduces the number of times the inner table must be read by an order of magnitude.

MySQL uses join buffering under these conditions:

- The join_buffer_size [415] system variable determines the size of each join buffer.

- Join buffering can be used when the join is of type ALL [569] or index [568] (in other words, when no possible keys can be used, and a full scan is done, of either the data or index rows, respectively), or range [568].
- One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
- A join buffer is never allocated for the first nonconst table, even if it would be of type ALL [569] or index [568].
- A join buffer is allocated prior to executing the join and freed after the query is done.
- Only columns of interest to the join are stored in the join buffer, not whole rows.

For the example join described previously for the NLJ algorithm (without buffering), the join would be done as follow using join buffering:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions,
          send to client
        }
      }
      empty buffer
    }
  }
}

if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions,
      send to client
    }
  }
}
```

If *S* is the size of each stored t1, t2 combination is the join buffer and *C* is the number of combinations in the buffer, the number of times table t3 is scanned is:

```
(S * C)/join_buffer_size + 1
```

One implication is that the number of t3 scans decreases as the value of join_buffer_size [415] increases, up to the point when join_buffer_size [415] is large enough to hold all previous row combinations. At that point, there is no speed to be gained by making it larger.

7.3.1.7 ORDER BY Optimization

In some cases, MySQL can use an index to satisfy an ORDER BY clause without doing any extra sorting.

The index can also be used even if the ORDER BY does not match the index exactly, as long as all of the unused portions of the index and all the extra ORDER BY columns are constants in the WHERE clause. The following queries use the index to resolve the ORDER BY part:

```
SELECT * FROM t1
  ORDER BY key_part1,key_part2,... ;

SELECT * FROM t1
  WHERE key_part1=constant
  ORDER BY key_part2;

SELECT * FROM t1
  ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
  WHERE key_part1=1
  ORDER BY key_part1 DESC, key_part2 DESC;
```

In some cases, MySQL *cannot* use indexes to resolve the ORDER BY, although it still uses indexes to find the rows that match the WHERE clause. These cases include the following:

- You use ORDER BY on different keys:

  ```
  SELECT * FROM t1 ORDER BY key1, key2;
  ```

- You use ORDER BY on nonconsecutive parts of a key:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
  ```

- You mix ASC and DESC:

  ```
  SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
  ```

- The key used to fetch the rows is not the same as the one used in the ORDER BY:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
  ```

- You use ORDER BY with an expression that includes terms other than the key column name:

  ```
  SELECT * FROM t1 ORDER BY ABS(key);
  SELECT * FROM t1 ORDER BY -key;
  ```

- You are joining many tables, and the columns in the ORDER BY are not all from the first nonconstant table that is used to retrieve rows. (This is the first table in the EXPLAIN output that does not have a const [567] join type.)

- You have different ORDER BY and GROUP BY expressions.

- You index only a prefix of a column named in the ORDER BY clause. In this case, the index cannot be used to fully resolve the sort order. For example, if you have a CHAR(20) column, but index only the first 10 bytes, the index cannot distinguish values past the 10th byte and a filesort will be needed.

- The type of table index used does not store rows in order. For example, this is true for a HASH index in a MEMORY table.

Availability of an index for sorting may be affected by the use of column aliases. Suppose that the column t1.a is indexed. In this statement, the name of the column in the select list is a. It refers to t1.a, so for the reference to a in the ORDER BY, the index can be used:

```
SELECT a FROM t1 ORDER BY a;
```

In this statement, the name of the column in the select list is also a, but it is the alias name. It refers to ABS(a), so for the reference to a in the ORDER BY, the index cannot be used:

```
SELECT ABS(a) AS a FROM t1 ORDER BY a;
```

In the following statement, the ORDER BY refers to a name that is not the name of a column in the select list. But there is a column in t1 named a, so the ORDER BY uses that, and the index can be used. (The resulting sort order may be completely different from the order for ABS(a), of course.)

```
SELECT ABS(a) AS b FROM t1 ORDER BY a;
```

By default, MySQL sorts all GROUP BY *col1, col2, ...* queries as if you specified ORDER BY *col1, col2, ...* in the query as well. If you include an explicit ORDER BY clause that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs. If a query includes GROUP BY but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying ORDER BY NULL. For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

With EXPLAIN SELECT ... ORDER BY, you can check whether MySQL can use indexes to resolve the query. It cannot if you see Using filesort in the Extra column. See Section 7.2.1, “Optimizing Queries with EXPLAIN”.

MySQL has two filesort algorithms for sorting and retrieving results. The original method uses only the ORDER BY columns. The modified method uses not just the ORDER BY columns, but all the columns used in the query.

The optimizer selects which filesort algorithm to use. Prior to MySQL 4.1, it uses the original algorithm. As of MySQL 4.1, it normally uses the modified algorithm except when BLOB or TEXT columns are involved, in which case it uses the original algorithm.

The original filesort algorithm works as follows:

1. Read all rows according to key or by table scanning. Rows that do not match the WHERE clause are skipped.
2. For each row, store a pair of values in a buffer (the sort key and the row pointer). The size of the buffer is the value of the sort_buffer_size [427] system variable.
3. When the buffer gets full, run a qsort (quicksort) on it and store the result in a temporary file. Save a pointer to the sorted block. (If all pairs fit into the sort buffer, no temporary file is created.)
4. Repeat the preceding steps until all rows have been read.
5. Do a multi-merge of up to MERGEBUFF (7) regions to one block in another temporary file. Repeat until all blocks from the first file are in the second file.
6. Repeat the following until there are fewer than MERGEBUFF2 (15) blocks left.
7. On the last multi-merge, only the pointer to the row (the last part of the sort key) is written to a result file.
8. Read the rows in sorted order by using the row pointers in the result file. To optimize this, we read in a big block of row pointers, sort them, and use them to read the rows in sorted order into a row buffer.

The size of the buffer is the value of the read_rnd_buffer_size [426] system variable. The code for this step is in the sql/records.cc source file.

One problem with this approach is that it reads rows twice: One time when evaluating the WHERE clause, and again after sorting the pair values. And even if the rows were accessed successively the first time (for example, if a table scan is done), the second time they are accessed randomly. (The sort keys are ordered, but the row positions are not.)

The modified filesort algorithm incorporates an optimization such that it records not only the sort key value and row position, but also the columns required for the query. This avoids reading the rows twice. The modified filesort algorithm works like this:

1. Read the rows that match the WHERE clause.
2. For each row, record a tuple of values consisting of the sort key value and row position, and also the columns required for the query.
3. Sort the tuples by sort key value
4. Retrieve the rows in sorted order, but read the required columns directly from the sorted tuples rather than by accessing the table a second time.

Using the modified filesort algorithm, the tuples are longer than the pairs used in the original method, and fewer of them fit in the sort buffer (the size of which is given by sort_buffer_size [427]). As a result, it is possible for the extra I/O to make the modified approach slower, not faster. To avoid a slowdown, the optimization is used only if the total size of the extra columns in the sort tuple does not exceed the value of the max_length_for_sort_data [420] system variable. (A symptom of setting the value of this variable too high is that you should see high disk activity and low CPU activity.)

For slow queries for which filesort is not used, you might try lowering max_length_for_sort_data [420] to a value that is appropriate to trigger a filesort.

If you want to increase ORDER BY speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, you can try the following strategies:

- Increase the size of the sort_buffer_size [427] variable.
- Increase the size of the read_rnd_buffer_size [426] variable.
- Use less RAM per row by declaring columns only as large as they need to be to hold the values stored in them. For example, CHAR(16) is better than CHAR(200) if values never exceed 16 characters.
- Change tmpdir [432] to point to a dedicated file system with large amounts of free space. Also, if you use MySQL 4.1 or later, this option accepts several paths that are used in round-robin fashion, so you you can use this feature to spread the load across several directories. Paths should be separated by colon characters (“:”) on Unix and semicolon characters (“;”) on Windows, NetWare, and OS/2. The paths should be for directories in file systems that are located on different *physical* disks, not different partitions on the same disk.

7.3.1.8 GROUP BY Optimization

The most general way to satisfy a GROUP BY clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and to avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for GROUP BY are that all GROUP BY columns reference attributes from the same index, and that the index stores its keys in order (for example, this is a

BTREE index, and not a HASH index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

The following section details how a GROUP BY query can be executed through index access. The method first performs a range scan, and then groups the resulting tuples.

In MySQL, GROUP BY is used for sorting, so the server may also apply ORDER BY optimizations to grouping. See Section 7.3.1.7, “ORDER BY Optimization”.

Tight Index Scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a loose index scan are not met, it still may be possible to avoid creation of temporary tables for GROUP BY queries. If there are range conditions in the WHERE clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the WHERE clause, or scans the whole index if there are no range conditions, we term it a *tight index scan*. With a tight index scan, the grouping operation is performed only after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there is a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the GROUP BY key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If we require sorting of the GROUP BY result, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

Assume that there is an index idx(c1,c2,c3) on table t1(c1,c2,c3,c4). The following queries do not work with the loose index scan access method described earlier, but still work with the tight index scan access method.

- There is a gap in the GROUP BY, but it is covered by the condition c2 = 'a':

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- The GROUP BY does not begin with the first part of the key, but there is a condition that provides a constant for that part:

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

7.3.1.9 DISTINCT Optimization

DISTINCT combined with ORDER BY needs a temporary table in many cases.

Because DISTINCT may use GROUP BY, you should be aware of how MySQL works with columns in ORDER BY or HAVING clauses that are not part of the selected columns. See Section 11.15.3, “GROUP BY and HAVING with Hidden Columns”.

In most cases, a DISTINCT clause can be considered as a special case of GROUP BY. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;
```

```
SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to GROUP BY queries can be also applied to queries with a DISTINCT clause. Thus, for more details on the optimization possibilities for DISTINCT queries, see Section 7.3.1.8, "GROUP BY Optimization".

When combining LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that t1 is used before t2 (which you can check with EXPLAIN), MySQL stops reading from t2 (for any particular row in t1) when it finds the first row in t2:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

7.3.1.10 LIMIT Optimization

In some cases, MySQL handles a query differently when you are using LIMIT *row_count* and not using HAVING:

- If you are selecting only a few rows with LIMIT, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you use LIMIT *row_count* with ORDER BY, MySQL ends the sorting as soon as it has found the first *row_count* rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the LIMIT clause must be selected, and most or all of them must be sorted, before it can be ascertained that the first *row_count* rows have been found. In either case, after the initial rows have been found, there is no need to sort any remainder of the result set, and MySQL does not do so.
- When combining LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.
- In some cases, a GROUP BY can be resolved by reading the key in order (or doing a sort on the key) and then calculating summaries until the key value changes. In this case, LIMIT *row_count* does not calculate any unnecessary GROUP BY values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using SQL_CALC_FOUND_ROWS.
- LIMIT 0 quickly returns an empty set. This can be useful for checking the validity of a query. When using one of the MySQL APIs, it can also be employed for obtaining the types of the result columns. (This trick does not work in the MySQL Monitor (the mysql program), which merely displays Empty set in such cases; you should instead use SHOW COLUMNS or DESCRIBE for this purpose.)
- When the server uses temporary tables to resolve the query, it uses the LIMIT *row_count* clause to calculate how much space is required.

7.3.1.11 How to Avoid Table Scans

The output from EXPLAIN shows ALL in the type column when MySQL uses a table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length.

- There are no usable restrictions in the ON or WHERE clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See Section 7.3.1.2, “WHERE Clause Optimization”.
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably will do many key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use ANALYZE TABLE *tbl_name* to update the key distributions for the scanned table. See Section 12.4.2.1, “ANALYZE TABLE Syntax”.
- Use FORCE INDEX for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
```

  See Section 12.2.7.2, “Index Hint Syntax”.
- Start mysqld with the --max-seeks-for-key=1000 [420] option or use SET max_seeks_for_key=1000 to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.3, “Server System Variables”.

7.3.2 Optimizing Non-SELECT Statements

7.3.2.1 Speed of INSERT Statements

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by log *N*, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use INSERT statements with multiple VALUES lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row INSERT statements. If you are adding data to a nonempty

table, you can tune the bulk_insert_buffer_size [408] variable to make data insertion even faster. See Section 5.1.3, “Server System Variables”.

- If multiple clients are inserting a lot of rows, you can get higher speed by using the INSERT DELAYED statement. See Section 12.2.4.2, “INSERT DELAYED Syntax”.
- For a MyISAM table, you can use concurrent inserts to add rows at the same time that SELECT statements are running, if there are no deleted rows in middle of the data file. See Section 7.6.3, “Concurrent Inserts”.
- When loading a table from a text file, use LOAD DATA INFILE. This is usually 20 times faster than using INSERT statements. See Section 12.2.5, “LOAD DATA INFILE Syntax”.
- With some extra work, it is possible to make LOAD DATA INFILE run even faster for a MyISAM table when the table has many indexes. Use the following procedure:

  1. Optionally create the table with CREATE TABLE.
  2. Execute a FLUSH TABLES statement or a mysqladmin flush-tables command.
  3. Use myisamchk --keys-used=0 -rq */path/to/db/tbl_name*. This removes all use of indexes for the table.
  4. Insert data into the table with LOAD DATA INFILE. This does not update any indexes and therefore is very fast.
  5. If you intend only to read from the table in the future, use myisampack to compress it. See Section 13.1.3.3, “Compressed Table Characteristics”.
  6. Re-create the indexes with myisamchk -rq */path/to/db/tbl_name*. This creates the index tree in memory before writing it to disk, which is much faster that updating the index during LOAD DATA INFILE because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.
  7. Execute a FLUSH TABLES statement or a mysqladmin flush-tables command.

  LOAD DATA INFILE performs the preceding optimization automatically if the MyISAM table into which you insert data is empty. The main difference between automatic optimization and using the procedure explicitly is that you can let myisamchk allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the LOAD DATA INFILE statement.

  As of MySQL 4.0, you can also disable or enable the nonunique indexes for a MyISAM table by using the following statements rather than myisamchk. If you use these statements, you can skip the FLUSH TABLE operations:

```
ALTER TABLE tbl_name DISABLE KEYS;
ALTER TABLE tbl_name ENABLE KEYS;
```

- To speed up INSERT operations that are performed with multiple statements for nontransactional tables, lock your tables:

```
LOCK TABLES a WRITE;
INSERT INTO a VALUES (1,23),(2,34),(4,33);
INSERT INTO a VALUES (8,26),(6,29);
...
UNLOCK TABLES;
```

This benefits performance because the index buffer is flushed to disk only once, after all INSERT statements have completed. Normally, there would be as many index buffer flushes as there are INSERT statements. Explicit locking statements are not needed if you can insert all rows with a single INSERT.

To obtain faster insertions for transactional tables, you should use START TRANSACTION and COMMIT instead of LOCK TABLES.

Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. Suppose that five clients attempt to perform inserts simultaneously as follows:

- Connection 1 does 1000 inserts
- Connections 2, 3, and 4 do 1 insert
- Connection 5 does 1000 inserts

If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

INSERT, UPDATE, and DELETE operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five successive inserts or updates. If you do very many successive inserts, you could do a LOCK TABLES followed by an UNLOCK TABLES once in a while (each 1,000 rows or so) to permit other threads to access the table. This would still result in a nice performance gain.

INSERT is still much slower for loading data than LOAD DATA INFILE, even when using the strategies just outlined.

- To increase performance for MyISAM tables, for both LOAD DATA INFILE and INSERT, enlarge the key cache by increasing the key_buffer_size [415] system variable. See Section 7.8.2, “Tuning Server Parameters”.

7.3.2.2 Speed of UPDATE Statements

An update statement is optimized like a SELECT query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a MyISAM table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use OPTIMIZE TABLE occasionally. See Section 12.4.2.5, “OPTIMIZE TABLE Syntax”.

7.3.2.3 Speed of DELETE Statements

The time required to delete individual rows is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the key_buffer_size [415] system variable. See Section 7.8.2, “Tuning Server Parameters”.

To delete all rows from a table, TRUNCATE TABLE *tbl_name* is faster than than DELETE FROM *tbl_name*. Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock. See Section 12.1.10, “TRUNCATE TABLE Syntax”.

7.3.2.4 Speed of REPAIR TABLE Statements

REPAIR TABLE for MyISAM tables is similar to using myisamchk for repair operations, and some of the same performance optimizations apply:

- myisamchck has variables that control memory allocation. You may be able to its improve performance by setting these variables, as described in Section 4.6.2.6, “myisamchk Memory Usage”.
- For REPAIR TABLE, the same principle applies, but because the repair is done by the server, you set server system variables instead of myisamchk variables. Also, In addition to setting memory-allocation variables, increasing the myisam_max_sort_file_size [421] system variable increases the likelihood that the repair will use the faster filesort method and avoid the slower repair by key cache method. Set the variable to the maximum file size for your system, after checking to be sure that there is enough free space to hold a copy of the table files. The free space must be available in the file system containing the original table files.

Suppose that a myisamchk table-repair operation is done using the following options to set its memory-allocation variables:

```
--key_buffer_size=128M --sort_buffer_size=256M
--read_buffer_size=64M --write_buffer_size=64M
```

Some of those myisamchk variables correspond to server system variables:

myisamchk Variable	System Variable
key_buffer_size	key_buffer_size [415]
sort_buffer_size	myisam_sort_buffer_size [421]
read_buffer_size	read_buffer_size [425]
write_buffer_size	none

Each of the server system variables can be set at runtime, and some of them (myisam_sort_buffer_size [421], read_buffer_size [425]) have a session value in addition to a global value. Setting a session value limits the effect of the change to your current session and does not affect other users. Changing a global-only variable (key_buffer_size [415], myisam_max_sort_file_size [421]) affects other users as well. For key_buffer_size [415], you must take into account that the buffer is shared with those users. For example, if you set the myisamchk key_buffer_size variable to 128MB, you could set the corresponding key_buffer_size [415] system variable larger than that (if it is not already set larger), to allow for key buffer use by activity in other sessions. However, changing the global key buffer size invalidates the buffer, causing increased disk I/O and slowdown for other sessions. An alternative that avoids this problem is to use a separate key cache, assign to it the indexes from the table to be repaired, and deallocate it when the repair is complete. See Section 7.5.1.2, “Multiple Key Caches”.

Based on the preceding remarks, a REPAIR TABLE operation can be done as follows to use settings similar to the myisamchk command. Here a separate 128MB key buffer is allocated and the file system is assumed to permit a file size of at least 100GB.

```
SET SESSION myisam_sort_buffer_size = 256*1024*1024;
SET SESSION read_buffer_size = 64*1024*1024;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
SET GLOBAL repair_cache.key_buffer_size = 128*1024*1024;
CACHE INDEX tbl_name IN repair_cache;
LOAD INDEX INTO CACHE tbl_name;
REPAIR TABLE tbl_name ;
```

```
SET GLOBAL repair_cache.key_buffer_size = 0;
```

If you intend to change a global variable but want to do so only for the duration of a `REPAIR TABLE` operation to minimally affect other users, save its value in a user variable and restore it afterward. For example:

```
SET @old_myisam_sort_buffer_size = @@global.myisam_max_sort_file_size;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
REPAIR TABLE tbl_name ;
SET GLOBAL myisam_max_sort_file_size = @old_myisam_max_sort_file_size;
```

The system variables that affect `REPAIR TABLE` can be set globally at server startup if you want the values to be in effect by default. For example, add these lines to the server `my.cnf` file:

```
[mysqld]
myisam_sort_buffer_size=256M
key_buffer_size=1G
myisam_max_sort_file_size=100G
```

These settings do not include `read_buffer_size` [425]. Setting `read_buffer_size` [425] globally to a large value does so for all sessions and can cause performance to suffer due to excessive memory allocation for a server with many simultaneous sessions.

7.3.3 Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- Use persistent connections to the database to avoid connection overhead. If you cannot use persistent connections and you are initiating many new connections to the database, you may want to change the value of the `thread_cache_size` [431] variable. See Section 7.8.2, “Tuning Server Parameters”.
- Always check whether all your queries really use the indexes that you have created in the tables. In MySQL, you can do this with the `EXPLAIN` statement. See Section 7.2.1, “Optimizing Queries with `EXPLAIN`”.
- Try to avoid complex `SELECT` queries on `MyISAM` tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.
- `MyISAM` supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, you should consider using the table in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` to defragment the table after you have deleted a lot of rows from it. See Section 7.6.3, “Concurrent Inserts”.
- To fix any compression issues that may have occurred with `ARCHIVE` tables, you can use `OPTIMIZE TABLE`. See Section 13.7, “The `ARCHIVE` Storage Engine”.
- Use `ALTER TABLE ... ORDER BY` *`expr1, expr2, ...`* if you usually retrieve rows in *`expr1, expr2, ...`* order. By using this option after extensive changes to the table, you may be able to get higher performance.
- In some cases, it may make sense to introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique, and indexed, it may be much faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
  WHERE hash_col=MD5(CONCAT(col1,col2))
```

```
  AND col1='constant' AND col2='constant';
```

- For MyISAM tables that change frequently, you should try to avoid all variable-length columns (VARCHAR, BLOB, and TEXT). The table uses dynamic row format if it includes even a single variable-length column. See Chapter 13, *Storage Engines*.

- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a MyISAM table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See Chapter 13, *Storage Engines*.

- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

  ```
  UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
  ```

  This is very important when you use MySQL storage engines such as MyISAM and ISAM that have only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.

- If you need to collect statistics from large log tables, use summary tables instead of scanning the entire log table. Maintaining the summaries should be much faster than trying to calculate statistics “live.” Regenerating new summary tables from the logs when things change (depending on business decisions) is faster than changing the running application.

- If possible, you should classify reports as “live” or as “statistical,” where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.

- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.

- In some cases, it is convenient to pack and store data into a BLOB column. In this case, you must provide code in your application to pack and unpack information, but this may save a lot of accesses at some stage. This is practical when you have data that does not conform well to a rows-and-columns table structure.

- Normally, you should try to keep all data nonredundant (observing what is referred to in database theory as *third normal form*). However, there may be situations in which it can be advantageous to duplicate information or create summary tables to gain more speed.

- UDFs (user-defined functions) may be a good way to get more performance for some tasks. See Section 18.2, “Adding New Functions to MySQL”, for more information.

- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. If your database system supports table locks, this should help to ensure that the index cache is only flushed once after all updates. You can also take advantage of MySQL's query cache to achieve similar results; see Section 7.5.3, “The MySQL Query Cache”.

- Use INSERT DELAYED when you do not need to know when your data is written. This reduces the overall insertion impact because many rows can be written with a single disk write.

- Use INSERT LOW_PRIORITY when you want to give SELECT statements higher priority than your inserts.

Use `SELECT HIGH_PRIORITY` to get retrievals that jump the queue. That is, the `SELECT` is executed even if there is another client waiting to do a write.

`LOW_PRIORITY` and `HIGH_PRIORITY` have an effect only for storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- Use multiple-row `INSERT` statements to store many rows with one SQL statement. Many SQL servers support this, including MySQL.
- Use `LOAD DATA INFILE` to load large amounts of data. This is faster than using `INSERT` statements.
- Use `AUTO_INCREMENT` columns so that each row in a table can be identified by a single unique value. unique values.
- Use `OPTIMIZE TABLE` once in a while to avoid fragmentation with dynamic-format `MyISAM` tables. See Section 13.1.3, “`MyISAM` Table Storage Formats”.
- Use `MEMORY` (`HEAP`) tables when possible to get more speed. See Section 13.4, “The `MEMORY` (`HEAP`) Storage Engine”. `MEMORY` tables are useful for noncritical data that is accessed often, such as information about the last displayed banner for users who don't have cookies enabled in their Web browser. User sessions are another alternative available in many Web application environments for handling volatile state data.
- With Web servers, images and other binary assets should normally be stored as files. That is, store only a reference to the file rather than the file itself in the database. Most Web servers are better at caching files than database contents, so using files is generally faster.
- Columns with identical information in different tables should be declared to have identical data types so that joins based on the corresponding columns will be faster. Before MySQL 3.23, you get slow joins otherwise.

  Try to keep column names simple. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, you should keep them shorter than 18 characters.
- If you need really high speed, you should take a look at the low-level interfaces for data storage that the different SQL servers support. For example, by accessing the MySQL `MyISAM` storage engine directly, you could get a speed increase of two to five times compared to using the SQL interface. To be able to do this, the data must be on the same server as the application, and usually it should only be accessed by one process (because external file locking is really slow). One could eliminate these problems by introducing low-level `MyISAM` commands in the MySQL server (this could be one easy way to get more performance if needed). By carefully designing the database interface, it should be quite easy to support this type of optimization.
- If you are using numeric data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you need not parse your text files to find line and column boundaries.
- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See Chapter 14, *Replication*.
- Declaring a `MyISAM` table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you should ensure that the table is okay by running the server

with the `--myisam-recover` [390] option, or by running `myisamchk` before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)

7.4 Optimization and Indexes

7.4.1 Column Indexes

All MySQL data types can be indexed. Use of indexes on the relevant columns is the best way to improve the performance of `SELECT` operations.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See Chapter 13, *Storage Engines*. All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

The `MyISAM` and (as of MySQL 4.0.14) `InnoDB` storage engines also support indexing on `BLOB` and `TEXT` columns. When indexing a `BLOB` or `TEXT` column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for `InnoDB` tables). (Before MySQL 4.1.2, the limit is 255 bytes for all tables.) Note that prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE` statements is interpreted as number of characters. *Be sure to take this into account when specifying a prefix length for a column that uses a multi-byte character set*.

As of MySQL 3.23.23, you can also create `FULLTEXT` indexes. They are used for full-text searches. Only the `MyISAM` storage engine supports `FULLTEXT` indexes and only for `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always takes place over the entire column and column prefix indexing is not supported. For details, see Section 11.9, “Full-Text Search Functions”.

As of MySQL 4.1.0, you can create indexes on spatial data types. Spatial indexes use R-trees. Currently, only `MyISAM` supports indexes on spatial types.

The `MEMORY` (`HEAP`) storage engine uses `HASH` indexes by default. It also supports `BTREE` indexes as of MySQL 4.1.0.

7.4.2 Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 16 columns. For certain data types, you can index a prefix of the column (see Section 7.4.1, “Column Indexes”).

A multiple-column index can be considered a sorted array containing values that are created by concatenating the values of the indexed columns.

MySQL uses multiple-column indexes in such a way that queries are fast when you specify a known quantity for the first column of the index in a `WHERE` clause, even if you do not specify values for the other columns.

Suppose that a table has the following specification:

```
CREATE TABLE test (
```

```
    id         INT NOT NULL,
    last_name  CHAR(30) NOT NULL,
    first_name CHAR(30) NOT NULL,
    PRIMARY KEY (id),
    INDEX name (last_name,first_name)
);
```

The `name` index is an index over the `last_name` and `first_name` columns. The index can be used for queries that specify values in a known range for `last_name`, or for both `last_name` and `first_name`. Therefore, the `name` index is used in the following queries:

```
SELECT * FROM test WHERE last_name='Widenius';

SELECT * FROM test
  WHERE last_name='Widenius' AND first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius'
  AND (first_name='Michael' OR first_name='Monty');

SELECT * FROM test
  WHERE last_name='Widenius'
  AND first_name >='M' AND first_name < 'N';
```

However, the `name` index is *not* used in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius' OR first_name='Michael';
```

The manner in which MySQL uses indexes to improve query performance is discussed further in Section 7.4.3, “How MySQL Uses Indexes”.

7.4.3 How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. If a table has 1,000 rows, this is at least 100 times faster than reading sequentially. If you need to access most of the rows, it is faster to read sequentially, because this minimizes disk seeks.

Most MySQL indexes (`PRIMARY KEY`, `UNIQUE`, `INDEX`, and `FULLTEXT`) are stored in B-trees. Exceptions are that indexes on spatial data types use R-trees, and that `MEMORY` (`HEAP`) tables support hash indexes.

Strings are automatically prefix- and end-space compressed. See Section 12.1.4, “`CREATE INDEX` Syntax”.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in `MEMORY` tables) are described at the end of this section.

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows.

- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

  Comparison of dissimilar columns may prevent use of indexes if values cannot be compared directly without conversion. Suppose that a numeric column is compared to a string column. For a given value such as `1` in the numeric column, it might compare equal to any number of values in the string column such as `'1'`, `' 1'`, `'00001'`, or `'01.e1'`. This rules out use of any indexes for the string column.

- To find the `MIN()` [884] or `MAX()` [884] value for a specific indexed column *`key_col`*. This is optimized by a preprocessor that checks whether you are using `WHERE key_part_N = constant` on all key parts that occur before *`key_col`* in the index. In this case, MySQL does a single key lookup for each `MIN()` [884] or `MAX()` [884] expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key_part2),MAX(key_part2)
  FROM tbl_name WHERE key_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable key (for example, `ORDER BY key_part1, key_part2`). If all key parts are followed by `DESC`, the key is read in reverse order. See Section 7.3.1.7, “`ORDER BY` Optimization”, and Section 7.3.1.8, “`GROUP BY` Optimization”.

- In some cases, a query can be optimized to retrieve values without consulting the data rows. If a query uses only columns from a table that are numeric and that form a leftmost prefix for some key, the selected values may be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
  WHERE key_part1=1
```

Suppose that you issue the following `SELECT` statement:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on `col1` and `col2`, the appropriate rows can be fetched directly. If separate single-column indexes exist on `col1` and `col2`, the optimizer tries to find the most restrictive index by deciding which index finds fewer rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to find rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`.

MySQL cannot use an index if the columns do not form a leftmost prefix of the index. Suppose that you have the `SELECT` statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1, col2, col3)`, only the first two queries use the index. The third and fourth queries do involve indexed columns, but `(col2)` and `(col2, col3)` are not leftmost prefixes of `(col1, col2, col3)`.

B-Tree Index Characteristics

A B-tree index can be used for column comparisons in expressions that use the = [782], > [783], >= [782], < [782], <= [782], or BETWEEN [783] operators. The index also can be used for LIKE [804] comparisons if the argument to LIKE [804] is a constant string that does not start with a wildcard character. For example, the following SELECT statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with `'Patrick' <= key_col < 'Patricl'` are considered. In the second statement, only rows with `'Pat' <= key_col < 'Pau'` are considered.

The following SELECT statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the LIKE [804] value begins with a wildcard character. In the second statement, the LIKE [804] value is not a constant.

MySQL 4.0 and later versions perform an additional LIKE [804] optimization. If you use `... LIKE '%string%'` and *`string`* is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using *`col_name`* `IS NULL` employs indexes if *`col_name`* is indexed.

Any index that does not span all AND [787] levels in the WHERE clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every AND [787] group.

The following WHERE clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
    /* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
    /* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These WHERE clauses do *not* use indexes:

```
    /* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

    /*  Index is not used in both parts of the WHERE clause  */
... WHERE index=1 OR A=10

    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses LIMIT to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash Index Characteristics

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the = or <=> operators (but are *very* fast). They are not used for comparison operators such as < that find a range of values.
- The optimizer cannot use a hash index to speed up ORDER BY operations. (This type of index cannot be used to search for the next entry in order.)
- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a MyISAM table to a hash-indexed MEMORY table.
- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

7.4.4 MyISAM Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how may rows must be read for each ref [567] access
- To estimate how many row a partial join will produce; that is, the number of rows that an operation of this form will produce:

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The SHOW INDEX statement displays a cardinality value based on *N*/*S*, where *N* is the number of rows in the table and *S* is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the <=> comparison operator, NULL is not treated differently from any other value: NULL <=> NULL, just as *N* <=> *N* for any other *N*.

However, for a join based on the = operator, NULL is different from non-NULL values: *expr1* = *expr2* is not true when *expr1* or *expr2* (or both) are NULL. This affects ref [567] accesses for comparisons of the form *tbl_name.key* = *expr*: MySQL will not access the table if the current value of *expr* is NULL, because the comparison cannot be true.

For = comparisons, it does not matter how many NULL values are in the table. For optimization purposes, the relevant value is the average size of the non-NULL value groups. However, MySQL does not currently enable that average size to be collected or used.

For MyISAM tables, you have some control over collection of table statistics by means of the myisam_stats_method [422] system variable. This variable has three possible values, which differ as follows:

- When myisam_stats_method [422] is nulls_equal, all NULL values are treated as identical (that is, they all form a single value group).

  If the NULL value group size is much higher than the average non-NULL value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-NULL values. Consequently, the nulls_equal method may cause the optimizer not to use the index for ref [567] accesses when it should.

- When myisam_stats_method [422] is nulls_unequal, NULL values are not considered the same. Instead, each NULL value forms a separate value group of size 1.

  If you have many NULL values, this method skews the average value group size downward. If the average non-NULL value group size is large, counting NULL values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-NULL values. Consequently, the nulls_unequal method may cause the optimizer to use this index for ref [567] lookups when other methods may be better.

- When myisam_stats_method [422] is nulls_ignored, NULL values are ignored.

If you tend to use many joins that use <=> rather than =, NULL values are not special in comparisons and one NULL is equal to another. In this case, nulls_equal is the appropriate statistics method.

The myisam_stats_method [422] system variable has global and session values. Setting the global value affects MyISAM statistics collection for all MyISAM tables. Setting the session value affects statistics collection only for the current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of myisam_stats_method [422].

To regenerate table statistics, you can use any of the following methods:

- Set myisam_stats_method [422], and then issue a CHECK TABLE statement
- Execute myisamchk --stats_method=*method_name* --analyze
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set myisam_stats_method [422] and issue an ANALYZE TABLE statement

Some caveats regarding the use of myisam_stats_method [422]:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some ALTER TABLE statements, for example.) If this happens, the statistics are collected using whatever value myisam_stats_method [422] has at the time. Thus, if you collect statistics using one method, but myisam_stats_method [422] is set to the other method when a table's statistics are collected automatically later, the other method will be used.
- There is no way to tell which method was used to generate statistics for a given MyISAM table.
- myisam_stats_method [422] applies only to MyISAM tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the nulls_equal method.

7.5 Buffering and Caching

MySQL uses several strategies that cache information in memory buffers to increase performance.

7.5.1 The MyISAM Key Cache

To minimize disk I/O, the MyISAM storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system file system cache.

This section first describes the basic operation of the MyISAM key cache. Then it discusses changes made in MySQL 4.1 that improve key cache performance and that enable you to better control cache operation:

- Access to the key cache no longer is serialized among threads. Multiple sessions can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

The key cache mechanism also is used for ISAM tables. However, the significance of this fact is on the wane. ISAM table use has been decreasing since MySQL 3.23 when MyISAM was introduced. MySQL 4.1 carries this trend further; the ISAM storage engine is disabled by default. (Subsequent MySQL release series have no support at all for ISAM.)

To control the size of the key cache, use the key_buffer_size [415] system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the key_buffer_size [415] value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native file system buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the MyISAM index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are nonleaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU (Least Recently Used)* strategy: When choosing a block for replacement, it selects the least recently used index block. To make this choice easier, the key cache module maintains all used blocks in a special list (*LRU chain*) ordered by time of use. When a block is accessed, it is the most recently used and is placed at the end of the list. When blocks need to be replaced, blocks at the beginning of the list are the least recently used and become the first candidates for eviction.

The InnoDB storage engine also uses an LRU algorithm, to manage its buffer pool. See Section 7.5.2, “The InnoDB Buffer Pool”.

7.5.1.1 Shared Key Cache Access

Prior to MySQL 4.1, access to the key cache is serialized: No two threads can access key cache buffers simultaneously. The server processes a request for an index block only after it has finished processing the previous request. As a result, a request for an index block not present in any key cache buffer blocks access by other threads while a buffer is being updated to contain the requested index block.

Starting from version 4.1.0, the server supports shared access to the key cache:

- A buffer that is not being updated can be accessed by multiple sessions.
- A buffer that is being updated causes sessions that need to use it to wait until the update is complete.
- Multiple sessions can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

7.5.1.2 Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among sessions entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL 4.1.1 also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given `MyISAM` table. By default, all `MyISAM` table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the `CACHE INDEX` statement (see Section 12.4.6.1, “`CACHE INDEX` Syntax”). For example, the following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

Note

If the server has been built with the `ISAM` storage engine enabled, `ISAM` tables use the key cache mechanism. However, `ISAM` indexes use only the default key cache and cannot be reassigned to a different cache.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a `SET GLOBAL` parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

Note that you cannot destroy the default key cache. Any attempt to do this will be ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;

mysql> SHOW VARIABLES LIKE 'key_buffer_size';
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8384512 |
+-----------------+---------+
```

Key cache variables are structured system variables that have a name and components. For `keycache1.key_buffer_size`, `keycache1` is the cache variable name and `key_buffer_size` [415] is the cache component. See Section 5.1.4.1, “Structured System Variables”, for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, you can use a strategy that involves three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not updated.
- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.
- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to nonleaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read in from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

The `CACHE INDEX` statement sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init-file` option that names a file containing `CACHE INDEX` statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

The statements in mysqld_init.sql are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to hot_cache and cold_cache:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

7.5.1.3 Midpoint Insertion Strategy

By default, the key cache management system of MySQL 4.1 uses a simple LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy*.

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sublist and a warm sublist. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not "too short," always containing at least key_cache_division_limit [416] percent of the key cache blocks. key_cache_division_limit [416] is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sublist. After a certain number of hits (accesses of the block), it is promoted to the hot sublist. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sublist is placed at the end of the list. The block then circulates within this sublist. If the block stays at the beginning of the sublist for a long enough time, it is demoted to the warm sublist. This time is determined by the value of the key_cache_age_threshold [416] component of the key cache.

The threshold value prescribes that, for a key cache containing *N* blocks, the block at the beginning of the hot sublist not accessed within the last `N * key_cache_age_threshold / 100` hits is to be moved to the beginning of the warm sublist. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sublist.

The midpoint insertion strategy enables you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the key_cache_division_limit [416] value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the key_cache_division_limit [416] set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sublist during an index scan operation as well.

7.5.1.4 Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its nonleaf nodes, it makes sense to preload the key cache with index blocks before starting to use it. Preloading enables you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the LOAD INDEX INTO CACHE statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables t1 and t2:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

The IGNORE LEAVES modifier causes only blocks for the nonleaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from t1, but only blocks for the nonleaf nodes from t2.

If an index has been assigned to a key cache using a CACHE INDEX statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

7.5.1.5 Key Cache Block Size

MySQL 4.1 introduces a new key_cache_block_size [416] variable on a per-key cache basis. This variable specifies the size of the block buffers for a key cache. It is intended to enable tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

To control the size of blocks in the .MYI index file of MyISAM tables, use the --myisam-block-size [390] option at server startup.

7.5.1.6 Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the key_buffer_size [415] or key_cache_block_size [416] key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native file system caching. File system caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of file system caching for the indexes ceases.

7.5.2 The InnoDB Buffer Pool

InnoDB maintains a buffer pool for caching data and indexes in memory. InnoDB manages the pool as a list, using a least recently used (LRU) algorithm incorporating a midpoint insertion strategy. When room is needed to add a new block to the pool, InnoDB evicts the least recently used block and adds the new

block to the middle of the list. The midpoint insertion strategy in effect causes the list to be treated as two sublists:

- At the head, a sublist of “new” (or “young”) blocks that have been recently used.
- At the tail, a sublist of “old” blocks that are less recently used.

As a result of the algorithm, the new sublist contains blocks that are heavily used by queries. The old sublist contains less-used blocks, and candidates for eviction are taken from this sublist.

The LRU algorithm operates as follows by default:

- 3/8 of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.
- When `InnoDB` reads a block into the buffer pool, it initially inserts it at the midpoint (the head of the old sublist). A block can be read in as a result of two types of read requests: Because it is required (for example, to satisfy query execution), or as part of read-ahead performed in anticipation that it will be required.
- The first access to a block in the old sublist makes it “young”, causing it to move to the head of the buffer pool (the head of the new sublist). If the block was read in because it was required, the first access occurs immediately and the block is made young. If the block was read in due to read-ahead, the first access does not occur immediately (and might not occur at all before the block is evicted).
- As long as no accesses occur for a block in the pool, it “ages” by moving toward the tail of the list. Blocks in both the new and old sublists age as other blocks are made new. Blocks in the old sublist also age as blocks are inserted at the midpoint. Eventually, a block that remains unused for long enough reaches the tail of the old sublist and is evicted.

In the default operation of the buffer pool, a block when read in is loaded at the midpoint and then moved immediately to the head of the new sublist as soon as an access occurs. In the case of a table scan (such as performed for a `mysqldump` operation), each block read by the scan ends up moving to the head of the new sublist because multiple rows are accessed from each block. This occurs even for a one-time scan, where the blocks are not otherwise used by other queries. Blocks may also be loaded by the read-ahead background thread and then moved to the head of the new sublist by a single access. These effects can be disadvantageous because they push blocks that are in heavy use by other queries out of the new sublist to the old sublist where they become subject to eviction.

The `innodb_buffer_pool_size` [1067] system variable specifies the size of the buffer pool. If your buffer pool is small and you have sufficient memory, making the pool larger can improve performance by reducing the amount of disk I/O needed as queries access `InnoDB` tables.

The `MyISAM` storage engine also uses an LRU algorithm, to manage its key cache. See Section 7.5.1, “The `MyISAM` Key Cache”.

7.5.3 The MySQL Query Cache

From version 4.0.1 on, MySQL Server features a query cache. When in use, the query cache stores the text of a `SELECT` statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again. The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client.

The query cache can be useful in an environment where you have tables that do not change very often and for which the server receives many identical queries. This is a typical situation for many Web servers that generate many dynamic pages based on database content.

The query cache does not return stale data. When tables are modified, any relevant entries in the query cache are flushed.

Note

The query cache does not work in an environment where you have multiple `mysqld` servers updating the same `MyISAM` tables.

Note

The query cache is not used for prepared statements. If you are using prepared statements, consider that these statements will not be satisfied by the query cache. See Section 17.6.7, “C API Prepared Statements”.

Some performance data for the query cache follows. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2×500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you are performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` [424] system variable to 0. By disabling the query cache code, there is no noticeable overhead. If you build MySQL from source, query cache capabilities can be excluded from the server entirely by invoking `configure` with the `--without-query-cache` option.

The query cache offers the potential for substantial performance improvement, but you should not assume that it will do so under all circumstances. With some query cache configurations or server workloads, you might actually see a performance decrease:

- Be cautious about sizing the query cache excessively large, which increases the overhead required to maintain the cache, possibly beyond the benefit of enabling it. Sizes in tens of megabytes are usually beneficial. Sizes in the hundreds of megabytes might not be.
- Server workload has a significant effect on query cache efficiency. A query mix consisting almost entirely of a fixed set of `SELECT` statements is much more likely to benefit from enabling the cache than a mix in which frequent `INSERT` statements cause continual invalidation of results in the cache. In some cases, a workaround is to use the `SQL_NO_CACHE` option to prevent results from even entering the cache for `SELECT` statements that use frequently modified tables. (See Section 7.5.3.2, “Query Cache `SELECT` Options”.)

To verify that enabling the query cache is beneficial, test the operation of your MySQL server with the cache enabled and disabled. Then retest periodically because query cache efficiency may change as server workload changes.

7.5.3.1 How the Query Cache Operates

This section describes how the query cache works when it is operational. Section 7.5.3.3, “Query Cache Configuration”, describes how to control whether it is operational.

Incoming queries are compared to those in the query cache before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be *exactly* the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

Because comparison of a query against those in the cache occurs before parsing, the cache is not used for queries of the following types:

- Prepared statements
- Queries that are a subquery of an outer query

Before a query result is fetched from the query cache, MySQL checks whether the user has `SELECT` privilege for all databases and tables involved. If this is not the case, the cached result is not used.

If a query result is returned from query cache, the server increments the `Qcache_hits` [453] status variable, not `Com_select`. See Section 7.5.3.4, “Query Cache Status and Maintenance”.

If a table changes, all cached queries that use the table become invalid and are removed from the cache. This includes queries that use `MERGE` tables that map to the changed table. A table can be changed by many types of statements, such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, `ALTER TABLE`, `DROP TABLE`, or `DROP DATABASE`.

In MySQL 4.0, the query cache is disabled within transactions (it does not return results). Beginning with MySQL 4.1.1, the query cache also works within transactions when using `InnoDB` tables.

A query that begins with a leading comment may be cached, but cannot be fetched from the cache.

The query cache works for `SELECT SQL_CALC_FOUND_ROWS ...` queries and stores a value that is returned by a following `SELECT FOUND_ROWS()` query. `FOUND_ROWS()` [872] returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache. The `SELECT FOUND_ROWS()` query itself cannot be cached.

A query cannot be cached if it contains any of the functions shown in the following table.

`BENCHMARK()` [870]	`CONNECTION_ID()` [872]	`CONVERT_TZ()` [828]
`CURDATE()` [828]	`CURRENT_DATE()` [829]	`CURRENT_TIME()` [829]
`CURRENT_TIMESTAMP()` [829]	`CURTIME()` [829]	`DATABASE()` [872]
`ENCRYPT()` [867] with one parameter	`FOUND_ROWS()` [872]	`GET_LOCK()` [877]
`LAST_INSERT_ID()` [874]	`LOAD_FILE()` [798]	`MASTER_POS_WAIT()` [879]
`NOW()` [837]	`RAND()` [822]	`RELEASE_LOCK()` [879]
`SYSDATE()` [840]	`UNIX_TIMESTAMP()` [841] with no parameters	`USER()` [876]
`UUID()` [879]		

A query also is not cached under these conditions:

- It refers to user-defined functions (UDFs).

- It refers to user variables.
- It refers to tables in the `mysql` system database.
- It is of any of the following forms:

```
SELECT ... LOCK IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

  The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See the MyODBC section of Chapter 17, *Connectors and APIs*.

  Statements within transactions that use `SERIALIZABLE` [976] isolation level also cannot be cached because they use `LOCK IN SHARE MODE` locking.

- It was issued as a prepared statement, even if no placeholders were employed. For example, the query used here is not cached:

```
char *my_sql_stmt = "SELECT a, b FROM table_c";
/* ... */
mysql_stmt_prepare(stmt, my_sql_stmt, strlen(my_sql_stmt));
```

  See Section 17.6.7, “C API Prepared Statements”.

- It uses `TEMPORARY` tables.
- It does not use any tables.
- It generates warnings.
- The user has a column-level privilege for any of the involved tables.

7.5.3.2 Query Cache `SELECT` Options

Two query cache-related options may be specified in `SELECT` statements:

- `SQL_CACHE`

  The query result is cached if it is cacheable and the value of the `query_cache_type` [424] system variable is `ON` or `DEMAND`.

- `SQL_NO_CACHE`

  The query result is not cached.

Examples:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

7.5.3.3 Query Cache Configuration

The `have_query_cache` [413] server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting `mysqld`. The query cache system variables all have names that begin with `query_cache_`. They are described briefly in Section 5.1.3, “Server System Variables”, with additional configuration information given here.

To set the size of the query cache, set the `query_cache_size` [424] system variable. Setting it to 0 disables the query cache. The default size is 0, so the query cache is disabled by default.

Note

When using the Windows Configuration Wizard to install or configure MySQL, the default value for `query_cache_size` [424] will be configured automatically for you based on the different configuration types available. When using the Windows Configuration Wizard, the query cache may be enabled (that is, set to a nonzero value) due to the selected configuration. The query cache is also controlled by the setting of the `query_cache_type` [424] variable. You should check the values of these variables as set in your `my.ini` file after configuration has taken place.

When you set `query_cache_size` [424] to a nonzero value, keep in mind that the query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value too small, you'll get a warning, as in this example:

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1282
Message: Query cache failed to set size 39936;
         new query cache size is 0

mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 41984 |
+------------------+-------+
```

For the query cache to actually be able to hold any query results, its size must be set larger:

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| query_cache_size | 999424 |
+------------------+--------+
1 row in set (0.00 sec)
```

The query_cache_size [424] value is aligned to the nearest 1024 byte block. The value reported may therefore be different from the value that you assign.

If the query cache size is greater than 0, the query_cache_type [424] variable influences how it works. This variable can be set to the following values:

- A value of 0 or OFF prevents caching or retrieval of cached results.
- A value of 1 or ON enables caching except of those statements that begin with SELECT SQL_NO_CACHE.
- A value of 2 or DEMAND causes caching of only those statements that begin with SELECT SQL_CACHE.

Setting the GLOBAL query_cache_type [424] value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the SESSION query_cache_type [424] value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

If you set query_cache_type [424] at server startup (rather than at runtime with a SET statement), only the numeric values are permitted.

To control the maximum size of individual query results that can be cached, set the query_cache_limit [424] system variable. The default value is 1MB.

When a query is to be cached, its result (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the query_cache_min_res_unit [424] system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of query_cache_min_res_unit [424]:

- The default value of query_cache_min_res_unit [424] is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free blocks. Fragmentation can force the query cache to prune (delete) queries from the cache due to lack of memory. In this case, you should decrease the value of query_cache_min_res_unit [424]. The number of free blocks and queries removed due to pruning are given by the values of the Qcache_free_blocks [453] and Qcache_lowmem_prunes [453] status variables.
- If most of your queries have large results (check the Qcache_total_blocks [454] and Qcache_queries_in_cache [454] status variables), you can increase performance by increasing query_cache_min_res_unit [424]. However, be careful to not make it too large (see the previous item).

query_cache_min_res_unit [424] is present as of MySQL 4.1.

7.5.3.4 Query Cache Status and Maintenance

To check whether the query cache is present in your MySQL server, use the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
```

```
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The `RESET QUERY CACHE` statement removes all query results from the query cache. The `FLUSH TABLES` statement also does this.

To monitor query cache performance, use `SHOW STATUS` to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| Qcache_free_blocks      | 36     |
| Qcache_free_memory      | 138488 |
| Qcache_hits             | 79570  |
| Qcache_inserts          | 27087  |
| Qcache_lowmem_prunes    | 3114   |
| Qcache_not_cached       | 22989  |
| Qcache_queries_in_cache | 415    |
| Qcache_total_blocks     | 912    |
+-------------------------+--------+
```

Descriptions of each of these variables are given in Section 5.1.5, “Server Status Variables”. Some uses for them are described here.

The total number of `SELECT` queries is given by this formula:

```
  Com_select
+ Qcache_hits
+ queries with errors found by parser
```

The `Com_select` value is given by this formula:

```
  Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during the column-privileges check
```

The query cache uses variable-length blocks, so `Qcache_total_blocks` [454] and `Qcache_free_blocks` [453] may indicate query cache memory fragmentation. After `FLUSH QUERY CACHE`, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one table block needs to be allocated.

The information provided by the `Qcache_lowmem_prunes` [453] status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in Section 7.5.3.3, “Query Cache Configuration”.

7.6 Locking Issues

MySQL manages contention for table contents using locking:

- Internal locking is performed within the MySQL server itself to manage contention for table contents by multiple threads. This type of locking is internal because it is performed entirely by the server and involves no other programs. See Section 7.6.1, “Internal Locking Methods”.
- External locking occurs when the server and other programs lock table files to coordinate among themselves which program can access the tables at which time. See Section 7.6.4, “External Locking”.

7.6.1 Internal Locking Methods

This section discusses internal locking; that is, locking performed within the MySQL server itself to manage contention for table contents by multiple sessions. This type of locking is internal because it is performed entirely by the server and involves no other programs. External locking occurs when the server and other programs lock table files to coordinate among themselves which program can access the tables at which time. See Section 7.6.4, “External Locking”.

MySQL uses table-level locking for `ISAM`, `MyISAM`, `MEMORY` (`HEAP`), and `MERGE` tables, page-level locking for `BDB` tables, and row-level locking for `InnoDB` tables.

In many cases, you can make an educated guess about which locking type is best for an application, but generally it is difficult to say that a given lock type is better than another. Everything depends on the application and different parts of an application may require different lock types.

To decide whether you want to use a storage engine with row-level locking, you should look at what your application does and what mix of select and update statements it uses. For example, most Web applications perform many selects, relatively few deletes, updates based mainly on key values, and inserts into a few specific tables. The base MySQL `MyISAM` setup is very well tuned for this.

Table locking in MySQL is deadlock-free for storage engines that use table-level locking. Deadlock avoidance is managed by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order.

MySQL grants table write locks as follows:

1. If there are no locks on the table, put a write lock on it.
2. Otherwise, put the lock request in the write lock queue.

MySQL grants table read locks as follows:

1. If there are no write locks on the table, put a read lock on it.
2. Otherwise, put the lock request in the read lock queue.

Table updates are given higher priority than table retrievals. Therefore, when a lock is released, the lock is made available to the requests in the write lock queue and then to the requests in the read lock queue. This ensures that updates to a table are not “starved” even if there is heavy `SELECT` activity for the table. However, if you have many updates for a table, `SELECT` statements wait until there are no more updates.

For information on altering the priority of reads and writes, see Section 7.6.2, “Table Locking Issues”.

Starting in MySQL 3.23.33, you can analyze the table lock contention on your system by checking the `Table_locks_immediate` [457] and `Table_locks_waited` [457] status variables, which indicate the number of times that requests for table locks could be granted immediately and the number that had to wait, respectively:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----------------------+---------+
```

```
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324   |
+-----------------------+---------+
```

As of MySQL 3.23.7 (3.23.25 for Windows), the MyISAM storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a MyISAM table has no free blocks in the middle of the data file, rows are always inserted at the end of the data file. In this case, you can freely mix concurrent INSERT and SELECT statements for a MyISAM table without locks. That is, you can insert rows into a MyISAM table at the same time other clients are reading from it. (Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are enabled again automatically when all holes have been filled with new data.)

If you acquire a table lock explicitly with LOCK TABLES, you can request a READ LOCAL lock rather than a READ lock to enable other sessions to perform concurrent inserts while you have the table locked.

To perform many INSERT and SELECT operations on a table real_table when concurrent inserts are not possible, you can insert rows into a temporary table temp_table and update the real table with the rows from the temporary table periodically. This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, temp_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM temp_table;
mysql> DELETE FROM temp_table;
mysql> UNLOCK TABLES;
```

InnoDB uses row locks and BDB uses page locks. Deadlocks are possible for these storage engines because they automatically acquire locks during the processing of SQL statements, not at the start of the transaction.

Advantages of row-level locking:

- Fewer lock conflicts when different sessions access different rows
- Fewer changes for rollbacks
- Possible to lock a single row for a long time

Disadvantages of row-level locking:

- Requires more memory than page-level or table-level locks
- Slower than page-level or table-level locks when used on a large part of the table because you must acquire many more locks
- Slower than other locks if you often do GROUP BY operations on a large part of the data or if you must scan the entire table frequently

Generally, table locks are superior to page-level or row-level locks in the following cases:

- Most statements for the table are reads
- Statements for the table are a mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- `SELECT` combined with concurrent `INSERT` statements, and very few `UPDATE` or `DELETE` statements
- Many scans or `GROUP BY` operations on the entire table without any writers

With higher-level locks, you can more easily tune applications by supporting locks of different types, because the lock overhead is less than for row-level locks.

Options other than row-level or page-level locking:

- Versioning (such as that used in MySQL for concurrent inserts) where it is possible to have one writer at the same time as many readers. This means that the database or table supports different views for the data depending on when access begins. Other common terms for this are “time travel,” “copy on write,” or “copy on demand.”
- Copy on demand is in many cases superior to page-level or row-level locking. However, in the worst case, it can use much more memory than using normal locks.
- Instead of using row-level locks, you can employ application-level locks, such as those provided by `GET_LOCK()` [877] and `RELEASE_LOCK()` [879] in MySQL. These are advisory locks, so they work only with applications that cooperate with each other. See Section 11.14, “Miscellaneous Functions”.

7.6.2 Table Locking Issues

To achieve a very high lock speed, MySQL uses table locking (instead of page, row, or column locking) for all storage engines except `InnoDB`, `BDB`, and `NDBCLUSTER`.

For `InnoDB` and `BDB` tables, MySQL uses table locking only if you explicitly lock the table with `LOCK TABLES`. For these storage engines, avoid using `LOCK TABLES` at all, because `InnoDB` uses automatic row-level locking and `BDB` uses page-level locking to ensure transaction isolation.

For large tables, table locking is often better than row locking, but there are some disadvantages:

- Table locking enables many sessions to read from a table at the same time, but if a session wants to write to a table, it must first get exclusive access. During the update, all other sessions that want to access this particular table must wait until the update is done.
- Table locking causes problems in cases such as when a session is waiting because the disk is full and free space needs to become available before the session can proceed. In this case, all sessions that want to access the problem table are also put in a waiting state until more disk space is made available.

Table locking is also disadvantageous under the following scenario:

- A session issues a `SELECT` that takes a long time to run.
- Another session then issues an `UPDATE` on the same table. This session waits until the `SELECT` is finished.
- Another session issues another `SELECT` statement on the same table. Because `UPDATE` has higher priority than `SELECT`, this `SELECT` waits for the `UPDATE` to finish, *after* waiting for the first `SELECT` to finish.

The following items describe some ways to avoid or reduce contention caused by table locking:

- Try to get the `SELECT` statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-updates` [389]. For storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`), this gives all statements that update (modify) a table

lower priority than SELECT statements. In this case, the second SELECT statement in the preceding scenario would execute before the UPDATE statement, and would not need to wait for the first SELECT to finish.

- To specify that all updates issued in a specific connection should be done with low priority, set the low_priority_updates [418] server system variable equal to 1.
- To give a specific INSERT, UPDATE, or DELETE statement lower priority, use the LOW_PRIORITY attribute.
- To give a specific SELECT statement higher priority, use the HIGH_PRIORITY attribute. See Section 12.2.7, “SELECT Syntax”.
- Starting from MySQL 3.23.7, you can start mysqld with a low value for the max_write_lock_count [421] system variable to force MySQL to temporarily elevate the priority of all SELECT statements that are waiting for a table after a specific number of inserts to the table occur. This permits READ locks after a certain number of WRITE locks.
- If you have problems with INSERT combined with SELECT, consider switching to MyISAM tables, which support concurrent SELECT and INSERT statements. (See Section 7.6.3, “Concurrent Inserts”.)
- If you mix inserts and deletes on the same table, INSERT DELAYED may be of great help. See Section 12.2.4.2, “INSERT DELAYED Syntax”.
- If you have problems with mixed SELECT and DELETE statements, the LIMIT option to DELETE may help. See Section 12.2.1, “DELETE Syntax”.
- Using SQL_BUFFER_RESULT with SELECT statements can help to make the duration of table locks shorter. See Section 12.2.7, “SELECT Syntax”.
- You could change the locking code in mysys/thr_lock.c to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

Here are some tips concerning table locks in MySQL:

- Concurrent users are not a problem if you do not mix updates with selects that need to examine many rows in the same table.
- You can use LOCK TABLES to increase speed, because many updates within a single lock is much faster than updating without locks. Splitting table contents into separate tables may also help.
- If you encounter speed problems with table locks in MySQL, you may be able to improve performance by converting some of your tables to InnoDB or BDB tables. See Section 13.2, “The InnoDB Storage Engine”, and Section 13.5, “The BDB (BerkeleyDB) Storage Engine”.

7.6.3 Concurrent Inserts

As of MySQL 3.23.7 (3.23.25 for Windows), the MyISAM storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a MyISAM table has no holes in the data file (deleted rows in the middle), an INSERT statement can be executed to add rows to the end of the table at the same time that SELECT statements are reading rows from the table. If there are multiple INSERT statements, they are queued and performed in sequence, concurrently with the SELECT statements. The results of a concurrent INSERT may not be visible immediately.

Concurrent inserts are enabled by default, but can be disabled by setting the concurrent_insert [409] system variable to 0.

Under circumstances where concurrent inserts can be used, there is seldom any need to use the `DELAYED` modifier for `INSERT` statements. See Section 12.2.4.2, “`INSERT DELAYED` Syntax”.

If you are using the update log or binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. See Section 5.3.4, “The Binary Log”. In addition, for those statements a read lock is placed on the selected-from table such that inserts into that table are blocked. The effect is that concurrent inserts for that table must wait as well.

With `LOAD DATA INFILE`, if you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other sessions can retrieve data from the table while `LOAD DATA` is executing. Use of the `CONCURRENT` option affects the performance of `LOAD DATA` a bit, even if no other session is using the table at the same time.

If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` [389] option if the server was started with that option. It also causes concurrent inserts not to be used.

For `LOCK TABLE`, the difference between `READ LOCAL` and `READ` is that `READ LOCAL` permits nonconflicting `INSERT` statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock.

7.6.4 External Locking

External locking is the use of file system locking to manage contention for database tables by multiple processes. External locking is used in situations where a single process such as the MySQL server cannot be assumed to be the only process that requires access to tables. Here are some examples:

- If you run multiple servers that use the same database directory (not recommended), each server must have external locking enabled.
- If you use `myisamchk` to perform table maintenance operations on `MyISAM` tables, you must either ensure that the server is not running, or that the server has external locking enabled so that it locks table files as necessary to coordinate with `myisamchk` for access to the tables. The same is true for use of `myisampack` to pack `MyISAM` tables.

  If the server is run with external locking enabled, you can use `myisamchk` at any time for read operations such a checking tables. In this case, if the server tries to update a table that `myisamchk` is using, the server will wait for `myisamchk` to finish before it continues.

  If you use `myisamchk` for write operations such as repairing or optimizing tables, or if you use `myisampack` to pack tables, you *must* always ensure that the `mysqld` server is not using the table. If you don't stop `mysqld`, you should at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

With external locking in effect, each process that requires access to a table acquires a file system lock for the table files before proceeding to access the table. If all necessary locks cannot be acquired, the process is blocked from accessing the table until the locks can be obtained (after the process that currently holds the locks releases them).

External locking affects server performance because the server must sometimes wait for other processes before it can access tables.

External locking is unnecessary if you run a single server to access a given data directory (which is the usual case) and if no other programs such as `myisamchk` need to modify tables while the server is

running. If you only *read* tables with other programs, external locking is not required, although `myisamchk` might report warnings if the server changes tables while `myisamchk` is reading them.

With external locking disabled, to use `myisamchk`, you must either stop the server while `myisamchk` executes or else lock and flush the tables before running `myisamchk`. (See Section 7.8.1, “System Factors and Startup Parameter Tuning”.) To avoid this requirement as of MySQL 3.23, use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables.

For `mysqld`, external locking is controlled by the value of the `skip_external_locking` [427] system variable. (Before MySQL 4.0.3, this variable is named `skip_locking`.) When this variable is enabled, external locking is disabled, and vice versa. From MySQL 4.0 on, external locking is disabled by default. Before MySQL 4.0, external locking is enabled by default on Linux or when MySQL is configured to use MIT-pthreads.

Use of external locking can be controlled at server startup by using the `--external-locking` [387] or `--skip-external-locking` [392] option. (Before MySQL 4.0.3, these options are named `--enable-locking` and `--skip-locking`.)

If you do use external locking option to enable updates to `MyISAM` tables from many MySQL processes, you must ensure that the following conditions are satisfied:

- You should not use the query cache for queries that use tables that are updated by another process.
- You should not start the server with the `--delay-key-write=ALL` [386] option or use the `DELAY_KEY_WRITE=1` table option for any shared tables. Otherwise, index corruption can occur.

The easiest way to satisfy these conditions is to always use `--external-locking` [387] together with `--delay-key-write=OFF` [386] and `--query-cache-size=0` [424]. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

7.7 Optimizing Database Structure

7.7.1 Make Your Data as Small as Possible

Design your tables to minimize their space on the disk. This can result in huge improvements by reducing the amount of data written to and read from disk. Smaller tables normally require less main memory while their contents are being actively processed during query execution. Any space reduction for table data also results in smaller indexes that can be processed faster.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application may give you a big performance gain. See Chapter 13, *Storage Engines*.

You can get better performance for a table and minimize storage space by using the techniques listed here:

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. `MEDIUMINT` is often a better choice than `INT` because a `MEDIUMINT` column uses 25% less space.
- Declare columns to be `NOT NULL` if possible. It makes everything faster and you save one bit per column. If you really need `NULL` in your application, you should definitely use it. Just avoid having it on all columns by default.
- For `MyISAM` tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size row format is used. This is faster but unfortunately may waste some space. See

Section 13.1.3, “MyISAM Table Storage Formats”. You can hint that you want to have fixed length rows even if you have VARCHAR columns with the CREATE TABLE option ROW_FORMAT=FIXED.

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient.
- Create only the indexes that you really need. Indexes are good for retrieval but bad when you need to store data quickly. If you access a table mostly by searching on a combination of columns, create an index on them. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, the first column in the index should be the one with the most duplicates to obtain better compression of the index.
- If it is very likely that a string column has a unique prefix on the first number of characters, it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see Section 12.1.4, “CREATE INDEX Syntax”). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See Section 7.8.2, “Tuning Server Parameters”.
- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.

7.7.2 How MySQL Opens and Closes Tables

When you execute a mysqladmin status command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The Open tables value of 12 can be somewhat puzzling if you have only six tables.

MySQL is multi-threaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client sessions having different states on the same table, the table is opened independently by each concurrent session. This uses additional memory but normally increases performance. With MyISAM tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all sessions.)

The table_cache [431], max_connections [419], and max_tmp_tables [420] system variables affect the maximum number of files the server keeps open. If you increase one or more of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems permit you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

table_cache [431] is related to max_connections [419]. For example, for 200 concurrent running connections, you should have a table cache size of at least 200 * *N*, where *N* is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the table_cache [431] setting. If table_cache [431] is set too high, MySQL may run out of file descriptors and refuse connections, fail to perform queries, and be very unreliable. You also have to take into account that the MyISAM storage engine needs two file descriptors for each unique open table. You can increase the number of file descriptors available to MySQL using the --open-files-limit [391] startup option to mysqld. See Section B.5.2.18, “'*File*' Not Found and Similar Errors”.

The cache of open tables is kept at a level of `table_cache` [431] entries. The default value is 64; this can be changed with the `--table_cache` [431] option to `mysqld`. Note that MySQL may temporarily open more tables than this to execute queries.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_cache` [431] entries and a table in the cache is no longer being used by any threads.
- When a table flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a `mysqladmin flush-tables` or `mysqladmin refresh` command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, beginning with the table least recently used.
- If a new table needs to be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary. When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A `MyISAM` table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any `MyISAM` table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER *tbl_name* OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER *tbl_name* CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See Section 12.2.3, “HANDLER Syntax”.

You can determine whether your table cache is too small by checking the `mysqld` status variable `Opened_tables` [453], which indicates the number of table-opening operations since the server started:

```
mysql> SHOW STATUS LIKE 'Opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
+---------------+-------+
```

If the value is very large or increases rapidly, even when you have not issued many `FLUSH TABLES` statements, you should increase the table cache size. See Section 5.1.3, “Server System Variables”, and Section 5.1.5, “Server Status Variables”.

7.7.3 Disadvantages of Creating Many Tables in the Same Database

If you have many `MyISAM` or `ISAM` tables in a database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by increasing the number of entries permitted in the table cache.

7.7.4 How MySQL Uses Internal Temporary Tables

In some cases, the server creates internal temporary tables while processing queries. Such a table can be held in memory and processed by the `MEMORY` storage engine, or stored on disk and processed by the `MyISAM` storage engine. The server may create a temporary table initially as an in-memory table, then convert it to an on-disk table if it becomes too large. Users have no direct control over when the server creates an internal temporary table or which storage engine the server uses to manage it.

Temporary tables can be created under conditions such as these:

- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- `DISTINCT` combined with `ORDER BY` may require a temporary table.
- If you use the `SQL_SMALL_RESULT` option, MySQL uses an in-memory temporary table, unless the query also contains elements (described later) that require on-disk storage.

To determine whether a query requires a temporary table, use `EXPLAIN` and check the `Extra` column to see whether it says `Using temporary` (see Section 7.2.1, “Optimizing Queries with `EXPLAIN`”).

Some conditions prevent the use of an in-memory temporary table, in which case the server uses an on-disk table instead:

- Presence of a `BLOB` or `TEXT` column in the table
- Presence of any column in a `GROUP BY` or `DISTINCT` clause larger than 512 bytes
- Presence of any column larger than 512 bytes in the `SELECT` list, if `UNION` or `UNION ALL` is used

If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the `tmp_table_size` [432] and `max_heap_table_size` [419] values. This differs from `MEMORY` tables explicitly created with `CREATE TABLE`: For such tables, only the `max_heap_table_size` [419] system variable determines how large the table is permitted to grow and there is no conversion to on-disk format.

When the server creates an internal temporary table (either in memory or on disk), it increments the `Created_tmp_tables` [451] status variable. If the server creates the table on disk (either initially or by converting an in-memory table) it increments the `Created_tmp_disk_tables` [450] status variable.

7.8 Optimizing the MySQL Server

7.8.1 System Factors and Startup Parameter Tuning

We start with system-level factors, because some of these decisions must be made very early to achieve large performance gains. In other cases, a quick look at this section may suffice. However, it is always nice to have a sense of how much can be gained by changing factors that apply at this level.

The operating system to use is very important. To get the best use of multiple-CPU machines, you should use Solaris (because its threads implementation works well) or Linux (because the 2.4 and later kernels have good SMP support). Note that older Linux kernels have a 2GB filesize limit by default. If you have such a kernel and a need for files larger than 2GB, you should get the Large File Support (LFS) patch for the ext2 file system. Other file systems such as ReiserFS and XFS do not have this 2GB limitation.

Before using MySQL in production, we advise you to test it on your intended platform.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.
- Use the `--skip-external-locking` [392] MySQL option to avoid external locking. This option is turned on by default as of MySQL 4.0. Before that, it is on by default when compiling with MIT-pthreads, because `flock()` is not fully supported by MIT-pthreads on all platforms. It is also on by default for Linux because Linux file locking is not yet safe.

  Note that disabling external locking does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run `myisamchk`. On some systems it is mandatory to disable external locking because it does not work, anyway.

  The only case in which you cannot disable external locking is when you run multiple MySQL *servers* (not clients) on the same data, or if you run `myisamchk` to check (not repair) a table without telling the server to flush and lock the tables first. Note that using multiple MySQL servers to access the same data concurrently is generally *not* recommended, except when using MySQL Cluster.

  The `LOCK TABLES` and `UNLOCK TABLES` statements use internal locking, so you can use them even if external locking is disabled.

7.8.2 Tuning Server Parameters

You can determine the default buffer sizes used by the `mysqld` server with this command (prior to MySQL 4.1, omit `--verbose` [395]):

```
shell> mysqld --verbose --help
```

This command produces a list of all `mysqld` options and configurable system variables. The output includes the default variable values and looks something like this:

```
back_log                current value: 5
bdb_cache_size          current value: 1048540
binlog_cache_size       current value: 32768
connect_timeout         current value: 5
delayed_insert_limit    current value: 100
delayed_insert_timeout  current value: 300
delayed_queue_size      current value: 1000
flush_time              current value: 0
interactive_timeout     current value: 28800
join_buffer_size        current value: 131072
key_buffer_size         current value: 1048540
long_query_time         current value: 10
lower_case_table_names  current value: 0
max_allowed_packet      current value: 1048576
max_binlog_cache_size   current value: 4294967295
max_connect_errors      current value: 10
max_connections         current value: 100
max_delayed_threads     current value: 20
max_heap_table_size     current value: 16777216
max_join_size           current value: 4294967295
max_sort_length         current value: 1024
max_tmp_tables          current value: 32
max_write_lock_count    current value: 4294967295
myisam_sort_buffer_size current value: 8388608
net_buffer_length       current value: 16384
net_read_timeout        current value: 30
net_retry_count         current value: 10
net_write_timeout       current value: 60
```

```
read_buffer_size          current value: 131072
read_rnd_buffer_size      current value: 262144
slow_launch_time          current value: 2
sort_buffer               current value: 2097116
table_cache               current value: 64
thread_concurrency        current value: 10
thread_stack              current value: 131072
tmp_table_size            current value: 1048576
wait_timeout              current value: 28800
```

For a `mysqld` server that is currently running, you can see the current values of its system variables by connecting to it and issuing this statement:

```
mysql> SHOW VARIABLES;
```

You can also see some statistical and status indicators for a running server by issuing this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also can be obtained using `mysqladmin`:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

For a full description of all system and status variables, see Section 5.1.3, “Server System Variables”, and Section 5.1.5, “Server Status Variables”.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you get better performance by giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` [415] and `table_cache` [431]. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations. The examples use the `mysqld_safe` script and use `--var_name=value` syntax to set the variable *`var_name`* to the value *`value`*. This syntax is available as of MySQL 4.0. For older versions of MySQL, take the following differences into account:

- Use `safe_mysqld` rather than `mysqld_safe`.
- Set variables using `--set-variable=var_name=value` or `-O var_name=value` syntax.
- For variable names that end in `_size`, you may need to specify them without `_size`. For example, the old name for `sort_buffer_size` [427] is `sort_buffer`. The old name for `read_buffer_size` [425] is `record_buffer`. To see which variables your version of the server recognizes, use `mysqld --help`.

If you have at least 256MB of memory and many tables and want maximum performance with a moderate number of clients, you should use something like this:

```
shell> mysqld_safe --key_buffer_size=64M --table_cache=256 \
          --sort_buffer_size=4M --read_buffer_size=1M &
```

If you have only 128MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

```
shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
```

If there are very many simultaneous connections, swapping problems may occur unless `mysqld` has been configured to use very little memory for each connection. `mysqld` performs better if you have enough memory for all connections.

With little memory and lots of connections, use something like this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
           --read_buffer_size=100K &
```

Or even this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
           --table_cache=32 --read_buffer_size=8K \
           --net_buffer_length=1K &
```

If you are performing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, you should increase the value of `read_rnd_buffer_size` [426] to speed up the reading of rows following sorting operations.

You can make use of the example option files included with your MySQL distribution; see Preconfigured Option Files.

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file.

To see the effects of a parameter change, do something like this (prior to MySQL 4.1, omit `--verbose` [395]):

```
shell> mysqld --key_buffer_size=32M --verbose --help
```

The variable values are listed near the end of the output. Make sure that the `--verbose` [395] and `--help` [384] options are last. Otherwise, the effect of any options listed after them on the command line are not reflected in the output.

For information on tuning the `InnoDB` storage engine, see Section 13.2.14.1, “`InnoDB` Performance Tuning Tips”.

7.8.3 How MySQL Uses Threads for Client Connections

Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

In this connection thread model, there are as many threads as there are clients currently connected, which has some disadvantages when server workload must scale to handle large numbers of connections. For example, thread creation and disposal becomes expensive. Also, each thread requires server and kernel resources, such as stack space. To accommodate a large number of simultaneous connections, the stack size per thread must be kept small, leading to a situation where it is either too small or the server consumes large amounts of memory. Exhaustion of other resources can occur as well, and scheduling overhead can become significant.

To control and monitor how the server manages threads that handle client connections, several system and status variables are relevant. (See Section 5.1.3, “Server System Variables”, and Section 5.1.5, “Server Status Variables”.)

The thread cache has a size determined by the `thread_cache_size` [431] system variable. The default value is 0 (no caching), which causes a thread to be set up for each new connection and disposed of when the connection terminates. Set `thread_cache_size` [431] to *`N`* to enable *`N`* inactive connection threads to be cached. `thread_cache_size` [431] can be set at server startup or changed while the server runs. A connection thread becomes inactive when the client connection with which it was associated terminates.

To monitor the number of threads in the cache and how many threads have been created because a thread could not be taken from the cache, monitor the `Threads_cached` [457] and `Threads_created` [457] status variables.

You can set `max_connections` [419] at server startup or at runtime to control the maximum number of clients that can connect simultaneously.

When the thread stack is too small, this limits the complexity of the SQL statements which the server can handle, the recursion depth of stored procedures, and other memory-consuming actions. To set a stack size of *`N`* bytes for each thread, start the server with `--thread_stack=`*`N`* [431].

7.8.4 How MySQL Uses Memory

The following list indicates some of the ways that the `mysqld` server uses memory. Where applicable, the name of the system variable relevant to the memory use is given:

- All threads share the `MyISAM` key buffer; its size is determined by the `key_buffer_size` [415] variable. Other buffers used by the server are allocated as needed. See Section 7.8.2, “Tuning Server Parameters”.
- Each thread that is used to manage client connections uses some thread-specific space. The following list indicates these and which variables control their size:
  - A stack (variable `thread_stack` [431])
  - A connection buffer (variable `net_buffer_length` [422])
  - A result buffer (variable `net_buffer_length` [422])

  The connection buffer and result buffer each begin with a size equal to `net_buffer_length` [422] bytes, but are dynamically enlarged up to `max_allowed_packet` [418] bytes as needed. The result buffer shrinks to `net_buffer_length` [422] bytes after each SQL statement. While a statement is running, a copy of the current statement string is also allocated.
- All threads share the same base memory.
- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.

- Only compressed `ISAM` and `MyISAM` tables are memory mapped. This is because the 32-bit memory space of 4GB is not large enough for most big tables. When systems with a 64-bit address space become more common, we may add general support for memory mapping.
- Each request that performs a sequential scan of a table allocates a *read buffer* (variable `read_buffer_size` [425]).
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* (variable `read_rnd_buffer_size` [426]) may be allocated to avoid disk seeks.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain `BLOB` columns are stored on disk.

  One problem before MySQL 3.23.2 is that if an internal in-memory temporary table becomes too large, the error `The table tbl_name is full` occurs. From 3.23.2 on, MySQL handles this automatically by changing the table from in-memory to on-disk format, to be handled by the `MyISAM` storage engine. To work around this problem for older servers, you can increase the temporary table size by setting the `tmp_table_size` [432] option to `mysqld`, or by setting the SQL option `sql_big_tables` in the client program. See Section 7.7.4, “How MySQL Uses Internal Temporary Tables”, and Section 5.1.3, “Server System Variables”.

  In MySQL 3.20, the maximum size of the temporary table is `record_buffer*16`; if you are using this version, you have to increase the value of `record_buffer`. You can also start `mysqld` with the `--big-tables` [384] option to always store temporary tables on disk. However, this affects the speed of many complicated queries.
- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See Section B.5.4.4, “Where MySQL Stores Temporary Files”.
- Almost all parsing and calculating is done in thread-local and reusable memory pools. No memory overhead is needed for small items, so the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings.
- For each `MyISAM` or `ISAM` table that is opened, the index file is opened once and the data file is opened once for each concurrently running thread. For each concurrent thread, a table structure, column structures for each column, and a buffer of size `3 * N` are allocated (where `N` is the maximum row length, not counting `BLOB` columns). A `BLOB` column requires five to eight bytes plus the length of the `BLOB` data. The `MyISAM` and `ISAM` storage engines maintain one extra row buffer for internal use.
- For each table having `BLOB` columns, a buffer is enlarged dynamically to read in larger `BLOB` values. If you scan a table, a buffer as large as the largest `BLOB` value is allocated.
- Handler structures for all in-use tables are saved in a cache and managed as a FIFO. The initial cache size is taken from the value of the `table_cache` [431] system variable. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See Section 7.7.2, “How MySQL Opens and Closes Tables”.
- A `FLUSH TABLES` statement or `mysqladmin flush-tables` command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. `FLUSH TABLES` does not return until all tables have been closed.
- The server caches information in memory as a result of `GRANT` statements. This memory is not released by the corresponding `REVOKE` statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.

`ps` and other system status programs may report that `mysqld` uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of `ps` counts the unused memory between stacks as used memory. To verify this, check available swap with `swap -s`. We test `mysqld` with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

7.8.5 How MySQL Uses DNS

When a new client connects to `mysqld`, `mysqld` spawns a new thread to handle the request. This thread first checks whether the host name is in the host name cache. If not, the thread attempts to resolve the host name:

- The thread takes the IP address and resolves it to a host name (using `gethostbyaddr()`). It then takes that host name and resolves it back to the IP address (using `gethostbyname()`) and compares to ensure it is the original IP address.
- If the operating system supports the thread-safe `gethostbyaddr_r()` and `gethostbyname_r()` calls, the thread uses them to perform host name resolution.
- If the operating system does not support the thread-safe calls, the thread locks a mutex and calls `gethostbyaddr()` and `gethostbyname()` instead. In this case, no other thread can resolve host names that are not in the host name cache until the first thread unlocks the mutex.

You can disable DNS host name lookups by starting `mysqld` with the `--skip-name-resolve` [393] option. However, in this case, you can use only IP addresses in the MySQL grant tables.

If you have a very slow DNS and many hosts, you can get more performance by either disabling DNS lookups with `--skip-name-resolve` [393] or by increasing the `HOST_CACHE_SIZE` define (default value: 128) and recompiling `mysqld`.

You can disable the host name cache by starting the server with the `--skip-host-cache` [392] option. To clear the host name cache, issue a `FLUSH HOSTS` statement or execute the `mysqladmin flush-hosts` command.

To disallow TCP/IP connections entirely, start `mysqld` with the `--skip-networking` [393] option.

7.9 Disk Issues

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:
  - Using symbolic links

    This means that, for `MyISAM` tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See Section 7.10, “Using Symbolic Links”.
  - Striping

    Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the *`N`*-th block on the (*`N MOD number_of_disks`*) disk, and so on. This

means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See Section 7.1.4, “Using Your Own Benchmarks”.

The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.

- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need 2 × *N* drives to hold *N* drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.
- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID *N* disk. RAID *N* can be a problem if you have many writes, due to the time required to update the parity bits.
- On Linux, you can get much more performance by using `hdparm` to configure your disk's interface. (Up to 100% under load is not uncommon.) The following `hdparm` options should be quite good for MySQL, and probably for many other applications:

  ```
  hdparm -m 16 -d 1
  ```

  Note that performance and reliability when using this command depend on your hardware, so we strongly suggest that you test your system thoroughly after using `hdparm`. Please consult the `hdparm` manual page for more information. If `hdparm` is not used wisely, file system corruption may result, so back up everything before experimenting!
- You can also set the parameters for the file system that the database uses:

  If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your file systems with the `-o noatime` option. That skips updates to the last access time in inodes on the file system, which avoids some disk seeks.

  On many operating systems, you can set a file system to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you more performance without sacrificing too much reliability. (This flag is on by default on Linux.)

7.10 Using Symbolic Links

You can move tables and databases from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disk.

The recommended way to do this is simply to symlink databases to a different disk. Symlink tables only as a last resort.

7.10.1 Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a symlink to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
```

```
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, or any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, problems are likely to occur.

If you really need to do this, you can change one of the source files. The file to modify depends on your version of MySQL. For MySQL 4.0 and up, look for the following statement in the `mysys/my_symlink.c` file:

```
if (!(MyFlags & MY_RESOLVE_LINK) ||
    (!lstat(filename,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

Before MySQL 4.0, look for this statement in the `mysys/mf_format.c` file:

```
if (flag & 32 || (!lstat(to,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

Change the statement to this:

```
if (1)
```

7.10.2 Using Symbolic Links for Tables on Unix

Before MySQL 4.0, you should not symlink tables unless you are *very* careful with them. The problem is that if you run `ALTER TABLE`, `REPAIR TABLE`, or `OPTIMIZE TABLE` on a symlinked table, the symlinks are removed and replaced by the original files. This happens because these statements work by creating a temporary file in the database directory and replacing the original file with the temporary file when the statement operation is complete.

You should not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). You can check whether your system supports symbolic links by issuing a `SHOW VARIABLES LIKE 'have_symlink'` statement.

In MySQL 4.0, symlinks are fully supported only for `MyISAM` tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links.

The handling of symbolic links for `MyISAM` tables in MySQL 4.0 works the following way:

- In the data directory, you always have the table format (`.frm`) file, the data (`.MYD`) file, and the index (`.MYI`) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The format file cannot.
- You can symlink the data file and the index file independently to different directories.
- You can instruct a running MySQL server to perform the symlinking by using the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See Section 12.1.5, “`CREATE TABLE` Syntax”. Alternatively, symlinking can be accomplished manually from the command line using `ln -s` if `mysqld` is not running.

Note

Beginning with MySQL 4.1.24, the path used with either or both of the `DATA DIRECTORY` and `INDEX DIRECTORY` options may not include the MySQL `data` directory. (Bug #32167)

- `myisamchk` does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located. The same is true for the `ALTER TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.

- 

  Note

  When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason why you should *not* run `mysqld` as the system `root` or permit system users to have write access to MySQL database directories.

- If you rename a table with `ALTER TABLE ... RENAME` or `RENAME TABLE` and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.

- If you use `ALTER TABLE ... RENAME` or `RENAME TABLE` to move a table to another database, the table is moved to the other database directory. If the table name changed, the symlinks in the new database directory are renamed to the new names and the data file and index file are renamed accordingly.

- If you are not using symlinks, you should use the `--skip-symbolic-links` [393] option to `mysqld` to ensure that no one can use `mysqld` to drop or rename a file outside of the data directory.

`SHOW CREATE TABLE` does not report if a table has symbolic links prior to MySQL 4.0.15. This is also true for `mysqldump`, which uses `SHOW CREATE TABLE` to generate `CREATE TABLE` statements.

Table symlink operations that are not supported up through MySQL 4.1:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.

- `BACKUP TABLE` and `RESTORE TABLE` do not respect symbolic links.

- The `.frm` file must *never* be a symbolic link (as indicated previously, only the data and index files can be symbolic links). Attempting to do this (for example, to make synonyms) produces incorrect results. Suppose that you have a database `db1` under the MySQL data directory, a table `tbl1` in this database, and in the `db1` directory you make a symlink `tbl2` that points to `tbl1`:

  ```
  shell> cd /path/to/datadir/db1
  shell> ln -s tbl1.frm tbl2.frm
  shell> ln -s tbl1.MYD tbl2.MYD
  shell> ln -s tbl1.MYI tbl2.MYI
  ```

  Problems result if one thread reads `db1.tbl1` and another thread updates `db1.tbl2`:

  - The query cache is “fooled” (it has no way of knowing that `tbl1` has not been updated, so it returns outdated results).

  - `ALTER` statements on `tbl2` fail.

7.10.3 Using Symbolic Links for Databases on Windows

Beginning with MySQL 3.23.16, the `mysqld-max` and `mysql-max-nt` servers for Windows are compiled with the `-DUSE_SYMDIR` option. This enables you to put a database directory on a different disk by setting up a symbolic link to it. This is similar to the way that symbolic links work on Unix, although the procedure for setting up the link is different.

It is necessary to define `USE_SYMDIR` explicitly only before MySQL 4.0; for `mysqld-max` and `mysql-max-nt`, you can enable symbolic links by using the `--symbolic-links` [393] option. As of MySQL 4.0, symbolic links are enabled by default for all Windows servers. If you do not need them, you can disable them with the `--skip-symbolic-links` [393] option.

On Windows, create a symbolic link to a MySQL database by creating a file in the data directory that contains the path to the destination directory. The file should be named *`db_name`*`.sym`, where *`db_name`* is the database name.

Suppose that the MySQL data directory is `C:\mysql\data` and you want to have database `foo` located at `D:\data\foo`. Set up a symlink using this procedure:

1. Make sure that the `D:\data\foo` directory exists by creating it if necessary. If you already have a database directory named `foo` in the data directory, you should move it to `D:\data`. Otherwise, the symbolic link will be ineffective. To avoid problems, make sure that the server is not running when you move the database directory.
2. Create a text file `C:\mysql\data\foo.sym` that contains the path name `D:\data\foo\`.

Note

The path name to the new database and tables should be absolute. If you specify a relative path, the location will be relative to the `foo.sym` file.

After this, all tables created in the database `foo` are created in `D:\data\foo`.

The following limitations apply to the use of `.sym` files for database symbolic linking on Windows:

- The symbolic link is not used if a directory with the same name as the database exists in the MySQL data directory.
- The `--innodb_file_per_table` [1068] option cannot be used.
- If you run `mysqld` as a service, you cannot use a mapped drive to a remote server as the destination of the symbolic link. As a workaround, you can use the full path (`\\servername\path\`).

7.11 Examining Thread Information

When you are attempting to ascertain what your MySQL server is doing, it can be helpful to examine the process list, which is the set of threads currently executing within the server. Process list information is available from these sources:

- The `SHOW [FULL] PROCESSLIST` statement: Section 12.4.5.19, “`SHOW PROCESSLIST` Syntax”
- The `mysqladmin processlist` command: Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

You can always view information about your own threads. To view information about threads being executed for other accounts, you must have the `PROCESS` [492] privilege.

Each process list entry contains several pieces of information:

- `Id` is the connection identifier for the client associated with the thread.

- `User` and `Host` indicate the account associated with the thread.
- `db` is the default database for the thread, or `NULL` if none is selected.
- `Command` and `State` indicate what the thread is doing.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.
- `Time` indicates how long the thread has been in its current state. The thread's notion of the current time may be altered in some cases: The thread can change the time with `SET TIMESTAMP = value`. For a thread running on a slave that is processing events from the master, the thread time is set to the time found in the events and thus reflects current time on the master and not the slave.
- `Info` contains the text of the statement being executed by the thread, or `NULL` if it is not executing one. By default, this value contains only the first 100 characters of the statement. To see the complete statements, use `SHOW FULL PROCESSLIST`.

The following sections list the possible `Command` values, and `State` values grouped by category. The meaning for some of these values is self-evident. For others, additional description is provided.

7.11.1 Thread Command Values

A thread can have any of the following `Command` values:

- `Binlog Dump`

  This is a thread on a master server for sending binary log contents to a slave server.
- `Change user`

  The thread is executing a change-user operation.
- `Close stmt`

  The thread is closing a prepared statement.
- `Connect`

  A replication slave is connected to its master.
- `Connect Out`

  A replication slave is connecting to its master.
- `Create DB`

  The thread is executing a create-database operation.
- `Daemon`

  This thread is internal to the server, not a thread that services a client connection.
- `Debug`

  The thread is generating debugging information.
- `Delayed insert`

The thread is a delayed-insert handler.

- `Drop DB`

  The thread is executing a drop-database operation.

- `Error`

- `Execute`

  The thread is executing a prepared statement.

- `Fetch`

  The thread is fetching the results from executing a prepared statement.

- `Field List`

  The thread is retrieving information for table columns.

- `Init DB`

  The thread is selecting a default database.

- `Kill`

  The thread is killing another thread.

- `Long Data`

  The thread is retrieving long data in the result of executing a prepared statement.

- `Ping`

  The thread is handling a server-ping request.

- `Prepare`

  The thread is preparing a prepared statement.

- `Processlist`

  The thread is producing information about server threads.

- `Query`

  The thread is executing a statement.

- `Quit`

  The thread is terminating.

- `Refresh`

  The thread is flushing table, logs, or caches, or resetting status variable or replication server information.

- `Register Slave`

  The thread is registering a slave server.

- `Reset stmt`

  The thread is resetting a prepared statement.

- `Set option`

  The thread is setting or resetting a client statement-execution option.

- `Shutdown`

  The thread is shutting down the server.

- `Sleep`

  The thread is waiting for the client to send a new statement to it.

- `Statistics`

  The thread is producing server-status information.

- `Table Dump`

  The thread is sending table contents to a slave server.

- `Time`

  Unused.

7.11.2 General Thread States

The following list describes thread `State` values that are associated with general query processing and not more specialized activities such as replication. Many of these are useful only for finding bugs in the server.

- `After create`

  This occurs when the thread creates a table (including internal temporary tables), at the end of the function that creates the table. This state is used even if the table could not be created due to some error.

- `Analyzing`

  The thread is calculating a `MyISAM` table key distributions (for example, for `ANALYZE TABLE`).

- `Checking table`

  The thread is performing a table check operation.

- `cleaning up`

  The thread has processed one command and is preparing to free memory and reset certain state variables.

- `closing tables`

  The thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, you should verify that you do not have a full disk and that the disk is not in very heavy use.

- `converting HEAP to MyISAM`

  The thread is converting an internal temporary table from a `MEMORY` table to an on-disk `MyISAM` table.

- `copy to tmp table`

  The thread is processing an `ALTER TABLE` statement. This state occurs after the table with the new structure has been created but before rows are copied into it.

- `Copying to group table`

  If a statement has different `ORDER BY` and `GROUP BY` criteria, the rows are sorted by group and copied to a temporary table.

- `Copying to tmp table`

  The server is copying to a temporary table in memory.

- `Copying to tmp table on disk`

  The server is copying to a temporary table on disk. The temporary result set has become too large (see Section 7.7.4, “How MySQL Uses Internal Temporary Tables”). Consequently, the thread is changing the temporary table from in-memory to disk-based format to save memory.

- `Creating index`

  The thread is processing `ALTER TABLE ... ENABLE KEYS` for a `MyISAM` table.

- `Creating sort index`

  The thread is processing a `SELECT` that is resolved using an internal temporary table.

- `creating table`

  The thread is creating a table. This includes creation of temporary tables.

- `Creating tmp table`

  The thread is creating a temporary table in memory or on disk. If the table is created in memory but later is converted to an on-disk table, the state during that operation will be `Copying to tmp table on disk`.

- `deleting from main table`

  The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving columns and offsets to be used for deleting from the other (reference) tables.

- `deleting from reference tables`

  The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- `discard_or_import_tablespace`

  The thread is processing an `ALTER TABLE ... DISCARD TABLESPACE` or `ALTER TABLE ... IMPORT TABLESPACE` statement.

- `end`

This occurs at the end but before the cleanup of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

- `executing`

  The thread has begun executing a statement.

- `Execution of init_command`

  The thread is executing statements in the value of the `init_command` system variable.

- `freeing items`

  The thread has executed a command. Some freeing of items done during this state involves the query cache. This state is usually followed by `cleaning up`.

- `Flushing tables`

  The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables.

- `FULLTEXT initialization`

  The server is preparing to perform a natural-language full-text search.

- `init`

  This occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements. Actions taken by the server in this state include flushing the binary log, the `InnoDB` log, and some query cache cleanup operations.

  For the `end` state, the following operations could be happening:

  - Removing query cache entries after data in a table is changed
  - Writing an event to the binary log
  - Freeing memory buffers, including for blobs

- `Killed`

  Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `Locked`

  The query is locked by another query.

- `logging slow query`

  The thread is writing a statement to the slow-query log.

- `NULL`

  This state is used for the `SHOW PROCESSLIST` state.

- `login`

The initial state for a connection thread until the client has been authenticated successfully.

- `Opening tables`, `Opening table`

  The thread is trying to open a table. This is should be very fast procedure, unless something prevents opening. For example, an `ALTER TABLE` or a `LOCK TABLE` statement can prevent opening a table until the statement is finished. It is also worth checking that your `table_cache` [431] value is large enough.

- `optimizing`

  The server is performing initial optimizations for a query.

- `preparing`

  This state occurs during query optimization.

- `Purging old relay logs`

  The thread is removing unneeded relay log files.

- `query end`

  This state occurs after processing a query but before the `freeing items` state.

- `Reading from net`

  The server is reading a packet from the network.

- `Removing duplicates`

  The query was using `SELECT DISTINCT` in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- `removing tmp table`

  The thread is removing an internal temporary table after processing a `SELECT` statement. This state is not used if no temporary table was created.

- `rename`

  The thread is renaming a table.

- `rename result table`

  The thread is processing an `ALTER TABLE` statement, has created the new table, and is renaming it to replace the original table.

- `Reopen tables`

  The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- `Repair by sorting`

  The repair code is using a sort to create indexes.

- `Repair done`

The thread has completed a multi-threaded repair for a `MyISAM` table.

- `Repair with keycache`

  The repair code is using creating keys one by one through the key cache. This is much slower than `Repair by sorting`.

- `Rolling back`

  The thread is rolling back a transaction.

- `Saving state`

  For `MyISAM` table operations such as repair or analysis, the thread is saving the new table state to the `.MYI` file header. State includes information such as number of rows, the `AUTO_INCREMENT` counter, and key distributions.

- `Searching rows for update`

  The thread is doing a first phase to find all matching rows before updating them. This has to be done if the `UPDATE` is changing the index that is used to find the involved rows.

- `Sending data`

  The thread is processing rows for a `SELECT` statement and also is sending data to the client.

- `setup`

  The thread is beginning an `ALTER TABLE` operation.

- `Sorting for group`

  The thread is doing a sort to satisfy a `GROUP BY`.

- `Sorting for order`

  The thread is doing a sort to satisfy a `ORDER BY`.

- `Sorting index`

  The thread is sorting index pages for more efficient access during a `MyISAM` table optimization operation.

- `Sorting result`

  For a `SELECT` statement, this is similar to `Creating sort index`, but for nontemporary tables.

- `statistics`

  The server is calculating statistics to develop a query execution plan. If a thread is in this state for a long time, the server is probably disk-bound performing other work.

- `System lock`

  The thread is going to request or is waiting for an internal or external system lock for the table. If this state is being caused by requests for external locks and you are not using multiple `mysqld` servers that are accessing the same tables, you can disable external system locks with the `--skip-external-locking` [392] option. However, external locking has been disabled by default since MySQL 4.0, so it is likely that this option will have no effect.

- `Table lock`

  The next thread state after `System lock`. The thread has acquired an external lock and is going to request an internal table lock.

- `update`

  The thread is getting ready to start updating the table.

- `Updating`

  The thread is searching for rows to update and is updating them.

- `updating main table`

  The server is executing the first part of a multiple-table update. It is updating only the first table, and saving columns and offsets to be used for updating the other (reference) tables.

- `updating reference tables`

  The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- `User lock`

  The thread is going to request or is waiting for an advisory lock requested with a `GET_LOCK()` [877] call.

- `Waiting for release of readlock`

  The thread is waiting for a global read lock obtained by another thread (with `FLUSH TABLES WITH READ LOCK`) to be released.

- `Waiting for tables`, `Waiting for table`

  The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

  This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES` *`tbl_name`*, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

- `Waiting on cond`

  A generic state in which the thread is waiting for a condition to become true. No specific state information is available.

- `Waiting to get readlock`

  The thread has issued a `FLUSH TABLES WITH READ LOCK` statement to obtain a global read lock and is waiting to obtain the lock.

- `Writing to net`

  The server is writing a packet to the network.

7.11.3 Delayed-Insert Thread States

These thread states are associated with processing for `DELAYED` inserts (see Section 12.2.4.2, “`INSERT DELAYED` Syntax”). Some states are associated with connection threads that process `INSERT DELAYED` statements from clients. Other states are associated with delayed-insert handler threads that insert the rows. There is a delayed-insert handler thread for each table for which `INSERT DELAYED` statements are issued.

States associated with a connection thread that processes an `INSERT DELAYED` statement from the client:

- `allocating local table`

  The thread is preparing to feed rows to the delayed-insert handler thread.

- `Creating delayed handler`

  The thread is creating a handler for `DELAYED` inserts.

- `got handler lock`

  This occurs before the `allocating local table` state and after the `waiting for handler lock` state, when the connection thread gets access to the delayed-insert handler thread.

- `got old table`

  This occurs after the `waiting for handler open` state. The delayed-insert handler thread has signaled that it has ended its initialization phase, which includes opening the table for delayed inserts.

- `storing row into queue`

  The thread is adding a new row to the list of rows that the delayed-insert handler thread must insert.

- `waiting for delay_list`

  This occurs during the initialization phase when the thread is trying to find the delayed-insert handler thread for the table, and before attempting to gain access to the list of delayed-insert threads.

- `waiting for handler insert`

  An `INSERT DELAYED` handler has processed all pending inserts and is waiting for new ones.

- `waiting for handler lock`

  This occurs before the `allocating local table` state when the connection thread waits for access to the delayed-insert handler thread.

- `waiting for handler open`

  This occurs after the `Creating delayed handler` state and before the `got old table` state. The delayed-insert handler thread has just been started, and the connection thread is waiting for it to initialize.

States associated with a delayed-insert handler thread that inserts the rows:

- `insert`

  The state that occurs just before inserting rows into the table.

- `reschedule`

After inserting a number of rows, the delayed-insert thread sleeps to let other threads do work.

- `upgrading lock`

  A delayed-insert handler is trying to get a lock for the table to insert rows.

- `Waiting for INSERT`

  A delayed-insert handler is waiting for a connection thread to add rows to the queue (see `storing row into queue`).

7.11.4 Replication Master Thread States

The following list shows the most common states you may see in the `State` column for the master's `Binlog Dump` thread. If you see no `Binlog Dump` threads on a master server, this means that replication is not running—that is, that no slaves are currently connected.

- `Sending binlog event to slave`

  Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the slave.

- `Finished reading one binlog; switching to next binlog`

  The thread has finished reading a binary log file and is opening the next one to send to the slave.

- `Has sent all binlog to slave; waiting for binlog to be updated`

  The thread has read all outstanding updates from the binary logs and sent them to the slave. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the master.

- `Waiting to finalize termination`

  A very brief state that occurs as the thread is stopping.

7.11.5 Replication Slave I/O Thread States

The following list shows the most common states you see in the `State` column for a slave server I/O thread. Beginning with MySQL 4.1.1, this state also appears in the `Slave_IO_State` column displayed by `SHOW SLAVE STATUS`, so you can get a good view of what is happening by using that statement.

- `Waiting for master update`

  The initial state before `Connecting to master`.

- `Connecting to master`

  The thread is attempting to connect to the master.

- `Checking master version`

  A state that occurs very briefly, after the connection to the master is established.

- `Registering slave on master`

  A state that occurs very briefly after the connection to the master is established.

- `Requesting binlog dump`

  A state that occurs very briefly, after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log file name and position.

- `Waiting to reconnect after a failed binlog dump request`

  If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `CHANGE MASTER TO` statement or the `--master-connect-retry` [1173] option.

- `Reconnecting after a failed binlog dump request`

  The thread is trying to reconnect to the master.

- `Waiting for master to send event`

  The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_net_timeout` [1181] seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- `Queueing master event to the relay log`

  The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- `Waiting to reconnect after a failed master event read`

  An error occurred while reading (due to disconnection). The thread is sleeping for the number of seconds set by the `CHANGE MASTER TO` statement or `--master-connect-retry` [1173] option (default 60) before attempting to reconnect.

- `Reconnecting after a failed master event read`

  The thread is trying to reconnect to the master. When connection is established again, the state becomes `Waiting for master to send event`.

- `Waiting for the slave SQL thread to free enough relay log space`

  You are using a nonzero `relay_log_space_limit` [426] value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- `Waiting for slave mutex on exit`

  A state that occurs briefly as the thread is stopping.

7.11.6 Replication Slave SQL Thread States

The following list shows the most common states you may see in the `State` column for a slave server SQL thread:

- `Waiting for the next event in relay log`

  The initial state before `Reading event from the relay log`.

- `Reading event from the relay log`

The thread has read an event from the relay log so that the event can be processed.

- `Has read all relay log; waiting for the slave I/O thread to update it`

  The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- `Making temp file`

  The thread is executing a `LOAD DATA INFILE` statement and is creating a temporary file containing the data from which the slave will read rows.

- `Waiting for slave mutex on exit`

  A very brief state that occurs as the thread is stopping.

The `State` column for the I/O thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and is executing it.

7.11.7 Replication Slave Connection Thread States

These thread states occur on a replication slave but are associated with connection threads, not with the I/O or SQL threads.

- `Changing master`

  The thread is processing a `CHANGE MASTER TO` statement.

- `Creating table from master dump`

  The slave is creating a table using the `CREATE TABLE` statement contained in the dump from the master. Used for `LOAD TABLE FROM MASTER` and `LOAD DATA FROM MASTER`.

- `Killing slave`

  The thread is processing a `SLAVE STOP` statement.

- `Opening master dump table`

  This state occurs after `Creating table from master dump`.

- `Reading master dump table data`

  This state occurs after `Opening master dump table`.

- `Rebuilding the index on master dump table`

  This state occurs after `Reading master dump table data`.

- `starting slave`

  The thread is starting the slave threads after processing a successful `LOAD DATA FROM MASTER` load operation.

7.11.8 MySQL Cluster Thread States

- `Committing events to binlog`

- `Opening mysql.ndb_apply_status`
- `Processing events`

  The thread is processing events for binary logging.

- `Processing events from schema table`

  The thread is doing the work of schema replication.

- `Shutting down`
- `Syncing ndb table schema operation and binlog`

  This is used to have a correct binary log of schema operations for NDB.

- `Waiting for event from ndbcluster`

  The server is acting as an SQL node in a MySQL Cluster, and is connected to a cluster management node.

- `Waiting for first event from ndbcluster`
- `Waiting for ndbcluster binlog update to reach current position`
- `Waiting for ndbcluster to start`
- `Waiting for schema epoch`

  The thread is waiting for a schema epoch (that is, a global checkpoint).

Chapter 8 Language Structure

Table of Contents

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- Reserved words
- User-defined and system variables
- Comments

8.1 Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal values, boolean values, and NULL. The section also covers the various nuances and “gotchas” that you may run into when dealing with these basic types in MySQL.

8.1.1 String Literals

A string is a sequence of bytes or characters, enclosed within either single quote (“'”) or double quote (“"”) characters. Examples:

```
'a string'
"another string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'
'a' ' ' 'string'
```

If the ANSI_QUOTES [458] SQL mode is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

As of MySQL 4.1.1, a *binary string* is a string of bytes that has no character set or collation. A *nonbinary string* is a string of characters that has a character set and collation. For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte. For nonbinary strings the unit is the character and some character sets support multi-byte characters. Character value ordering is a function of the string collation.

Also as of MySQL 4.1.1, string literals may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For more information about these forms of string syntax, see Section 9.1.3.5, “Character String Literal Character Set and Collation”, and Section 9.1.3.6, “National Character Set”.

Within a string, certain sequences have special meaning. Each of these sequences begins with a backslash (“`\`”), known as the *escape character*. MySQL recognizes the following escape sequences.

Character	Escape Sequence
`\0`	An ASCII NUL (`0x00`) character.
`\'`	A single quote (“`'`”) character.
`\"`	A double quote (“`"`”) character.
`\b`	A backspace character.
`\n`	A newline (linefeed) character.
`\r`	A carriage return character.
`\t`	A tab character.
`\Z`	ASCII 26 (Control-Z). See note following the table.
`\\`	A backslash (“`\`”) character.
`\%`	A “`%`” character. See note following the table.
`\_`	A “`_`” character. See note following the table.

For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, “`\x`” is just “`x`”.

These sequences are case sensitive. For example, “`\b`” is interpreted as a backspace, but “`\B`” is interpreted as “`B`”.

The ASCII 26 character can be encoded as “`\Z`” to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql db_name < file_name`.

Escape processing is done according to the character set indicated by the `character_set_connection` [408] system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in Section 9.1.3.5, “Character String Literal Character Set and Collation”.

The “`\%`” and “`\_`” sequences are used to search for literal instances of “`%`” and “`_`” in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the `LIKE` [804] operator in Section 11.5.1, “String Comparison Functions”. If you use “`\%`” or “`\_`” outside of pattern-matching contexts, they evaluate to the strings “`\%`” and “`\_`”, not to “`%`” and “`_`”.

There are several ways to include quote characters within a string:

- A “`'`” inside a string quoted with “`'`” may be written as “`''`”.
- A “`"`” inside a string quoted with “`"`” may be written as “`""`”.
- Precede the quote character by an escape character (“`\`”).
- A “`'`” inside a string quoted with “`"`” needs no special treatment and need not be doubled or escaped. In the same way, “`"`” inside a string quoted with “`'`” needs no special treatment.

The following `SELECT` statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';
+--------------------+
| This
Is
Four
Lines |
+--------------------+

mysql> SELECT 'disappearing\ backslash';
+------------------------+
| disappearing backslash |
+------------------------+
```

If you want to insert binary data into a string column (such as a `BLOB` column), the following characters must be represented by escape sequences.

Character	Escape Sequence
`NUL`	`NUL` byte (`0x00`). Represent this character by “`\0`” (a backslash followed by an ASCII “`0`” character).
`\`	Backslash (ASCII 92). Represent this character by “`\\`”.
`'`	Single quote (ASCII 39). Represent this character by “`\'`”.
`"`	Double quote (ASCII 34). Represent this character by “`\"`”.

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See Section 17.6.6.51, “`mysql_real_escape_string()`”. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See Section 17.8, “MySQL Perl API”. Other language interfaces may provide a similar capability.
- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

8.1.2 Number Literals

Integers are represented as a sequence of digits. Floats use “`.`” as a decimal separator. Either type of number may be preceded by “`-`” or “`+`” to indicate a negative or positive value, respectively

Examples of valid integers:

```
1221
0
-32
```

Examples of valid floating-point numbers:

```
294.42
-32032.6809e+10
148.00
```

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

8.1.3 Date and Time Literals

Date and time values can be represented in several formats, such as quoted strings or as numbers, depending on the exact type of the value and other factors. For example, in contexts where MySQL expects a date, it interprets any of `'2015-07-21'`, `'20150721'`, and `20150721` as a date.

This section describes the acceptable formats for date and time literals. For more information about the temporal data types, such as the range of permitted values, consult these sections:

- Section 10.1.2, “Date and Time Type Overview”
- Section 10.3, “Date and Time Types”

Standard SQL and ODBC Date and Time Literals

Standard SQL permits temporal literals to be specified using a type keyword and a string. The space between the keyword and string is optional.

```
DATE 'str'
TIME 'str'
TIMESTAMP 'str'
```

MySQL recognizes those constructions and also the corresponding ODBC syntax:

```
{ d 'str' }
{ t 'str' }
{ ts 'str' }
```

However, MySQL ignores the type keyword and each of the preceding constructions produces the string value `'str'`, with a type of `VARCHAR`.

String and Numeric Literals in Date and Time Context

MySQL recognizes `DATE` values in these formats:

- As a string in either `'YYYY-MM-DD'` or `'YY-MM-DD'` format. A “relaxed” syntax is permitted: Any punctuation character may be used as the delimiter between date parts. For example, `'2012-12-31'`, `'2012/12/31'`, `'2012^12^31'`, and `'2012@12@31'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDD'` or `'YYMMDD'` format, provided that the string makes sense as a date. For example, `'20070523'` and `'070523'` are interpreted as `'2007-05-23'`, but `'071332'` is illegal (it has nonsensical month and day parts) and becomes `'0000-00-00'`.
- As a number in either `YYYYMMDD` or `YYMMDD` format, provided that the number makes sense as a date. For example, `19830905` and `830905` are interpreted as `'1983-09-05'`.

MySQL recognizes `DATETIME` and `TIMESTAMP` values in these formats:

- As a string in either `'YYYY-MM-DD HH:MM:SS'` or `'YY-MM-DD HH:MM:SS'` format. A “relaxed” syntax is permitted here, too: Any punctuation character may be used as the delimiter between date parts or time parts. For example, `'2012-12-31 11:30:45'`, `'2012^12^31 11+30+45'`, `'2012/12/31 11*30*45'`, and `'2012@12@31 11^30^45'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDDHHMMSS'` or `'YYMMDDHHMMSS'` format, provided that the string makes sense as a date. For example, `'20070523091528'` and `'070523091528'` are interpreted as `'2007-05-23 09:15:28'`, but `'071122129015'` is illegal (it has a nonsensical minute part) and becomes `'0000-00-00 00:00:00'`.
- As a number in either `YYYYMMDDHHMMSS` or `YYMMDDHHMMSS` format, provided that the number makes sense as a date. For example, `19830905132800` and `830905132800` are interpreted as `'1983-09-05 13:28:00'`.

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `DATETIME` or `TIMESTAMP` columns. For information about fractional seconds support in MySQL, see Section 10.3.4, “Fractional Seconds in Time Values”.

Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

- Year values in the range `70-99` are converted to `1970-1999`.
- Year values in the range `00-69` are converted to `2000-2069`.

See also Section 10.3.6, “Two-Digit Years in Dates”.

For values specified as strings that include date part delimiters, it is unnecessary to specify two digits for month or day values that are less than `10`. `'2015-6-9'` is the same as `'2015-06-09'`. Similarly, for values specified as strings that include time part delimiters, it is unnecessary to specify two digits for hour, minute, or second values that are less than `10`. `'2015-10-30 1:2:3'` is the same as `'2015-10-30 01:02:03'`.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in `YYYYMMDD` or `YYYYMMDDHHMMSS` format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in `YYMMDD` or `YYMMDDHHMMSS` format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as nondelimited strings are interpreted according their length. For a string 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify `'9903'`, thinking that represents March, 1999, MySQL converts it to the “zero” date value. This occurs because the year and month values are `99` and `03`, but the day part is completely missing. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, to insert the value `'1999-03-00'`, use `'990300'`.

MySQL recognizes `TIME` values in these formats:

- As a string in `'D HH:MM:SS'` format. You can also use one of the following “relaxed” syntaxes: `'HH:MM:SS'`, `'HH:MM'`, `'D HH:MM'`, `'D HH'`, or `'SS'`. Here `D` represents days and can have a value from 0 to 34.
- As a string with no delimiters in `'HHMMSS'` format, provided that it makes sense as a time. For example, `'101112'` is understood as `'10:11:12'`, but `'109712'` is illegal (it has a nonsensical minute part) and becomes `'00:00:00'`.
- As a number in `HHMMSS` format, provided that it makes sense as a time. For example, `101112` is understood as `'10:11:12'`. The following alternative formats are also understood: `SS`, `MMSS`, or `HHMMSS`.

A trailing fractional seconds part is recognized in the `'D HH:MM:SS.fraction'`, `'HH:MM:SS.fraction'`, `'HHMMSS.fraction'`, and `HHMMSS.fraction` time formats, where `fraction` is the fractional part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into `TIME` columns. For information about fractional seconds support in MySQL, see Section 10.3.4, “Fractional Seconds in Time Values”.

For `TIME` values specified as strings that include a time part delimiter, it is unnecessary to specify two digits for hours, minutes, or seconds values that are less than `10`. `'8:3:2'` is the same as `'08:03:02'`.

8.1.4 Hexadecimal Literals

MySQL supports hexadecimal values, written using `X'val'`, `x'val'`, or `0xval` format, where *`val`* contains hexadecimal digits (`0..9`, `A..F`). Lettercase of the digits does not matter. For values written using `X'val'` or `x'val'` format, *`val`* must contain an even number of digits. For values written using `0xval` syntax, values that contain an odd number of digits are treated as having an extra leading `0`. For example, `0x0a` and `0xaaa` are interpreted as `0x0a` and `0x0aaa`.

In numeric contexts, hexadecimal values act like integers (64-bit precision). In string contexts, they act like binary strings, where each pair of hex digits is converted to a character:

```
mysql> SELECT X'4D7953514C';
        -> 'MySQL'
mysql> SELECT 0x0a+0;
        -> 10
mysql> SELECT 0x5061756c;
        -> 'Paul'
```

In MySQL 4.1 (and in MySQL 4.0 when using the `--new` option), the default type of a hexadecimal value is a string. If you want to ensure that the value is treated as a number, you can use `CAST(... AS UNSIGNED)` [859]:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
        -> 'A', 65
```

The `X'hexstring'` syntax is new in 4.0 and is based on standard SQL. The `0x` syntax is based on ODBC. Hexadecimal strings are often used by ODBC to supply values for `BLOB` columns.

Beginning with MySQL 4.0.1, you can convert a string or a number to a string in hexadecimal format with the `HEX()` [796] function:

```
mysql> SELECT HEX('cat');
        -> '636174'
mysql> SELECT 0x636174;
        -> 'cat'
```

8.1.5 Boolean Literals

Beginning with MySQL 4.1, The constants `TRUE` and `FALSE` evaluate to `1` and `0`, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
        -> 1, 1, 0, 0
```

8.1.6 `NULL` Values

The `NULL` value means “no data.” `NULL` can be written in any lettercase. A synonym is `\N` (case sensitive).

For text file import or export operations performed with `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, `NULL` is represented by the `\N` sequence. See Section 12.2.5, “`LOAD DATA INFILE` Syntax”.

Be aware that the `NULL` value is different from values such as `0` for numeric types or the empty string for string types. For more information, see Section B.5.5.3, “Problems with `NULL` Values”.

8.2 Database, Table, Index, Column, and Alias Names

Database, table, index, column, and alias names are identifiers. This section describes the permissible syntax for identifiers in MySQL. Section 8.2.2, “Identifier Case Sensitivity”, describes which types of identifiers are case sensitive and under what conditions.

An identifier may be quoted or unquoted. If an identifier contains special characters or is a reserved word, you *must* quote it whenever you refer to it. The set of alphanumeric characters from the current character set, “_”, and “$” are not special. Reserved words are listed at Section 8.3, “Reserved Words”. (Exception: A reserved word that follows a period in a qualified name must be an identifier, so it need not be quoted.)

The identifier quote character is the backtick (“`”):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the `ANSI_QUOTES` [458] SQL mode is enabled, it is also permissible to quote identifiers within double quotation marks:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax...
```

```
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

The ANSI_QUOTES [458] mode causes the server to interpret double-quoted strings as identifiers. Consequently, when this mode is enabled, string literals must be enclosed within single quotation marks. They cannot be enclosed within double quotation marks. The server SQL mode is controlled as described in Section 5.1.6, “Server SQL Modes”.

As of MySQL 4.1, identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, double the character. The following statement creates a table named a`b that contains a column named c"d:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
mysql> SELECT 1 AS `one`, 2 AS 'two';
+-----+-----+
| one | two |
+-----+-----+
|   1 |   2 |
+-----+-----+
```

Identifier quoting was introduced in MySQL 3.23.6 to permit use of identifiers that contain special characters or are reserved words. Before 3.23.6, you cannot use identifiers that require quotation marks, so the only legal characters are the set of alphanumeric characters from the current character set, “_”, and “$”.

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal.

Identifiers may begin with a digit but unless quoted may not consist solely of digits.

It is recommended that you do not use names that begin with *Me* or *MeN*, where *M* and *N* are integers. For example, avoid using 1e as an identifier, because an expression such as 1e+3 is ambiguous. Depending on context, it might be interpreted as the expression 1e + 3 or as the number 1e+3.

Be careful when using MD5() [868] to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See Section 8.4, “User-Defined Variables”, for more information and examples of workarounds.

There are some restrictions on the characters that may appear in identifiers:

- No identifier can contain ASCII NUL (0x00) or a byte with a value of 255.
- Before MySQL 4.1, identifier quote characters should not be used in identifiers.
- Database, table, and column names should not end with space characters.
- Database and table names cannot contain “/”, “\”, “.”, or characters that are not permitted in file names.

The following table describes the maximum length for each type of identifier. Before MySQL 4.1.5, the maximum-length restrictions on identifiers are measured in bytes, not characters. Until that version, if you

use multi-byte characters in your identifier names, the maximum length will depend on the byte count of all the characters used.

Identifier	Maximum Length
Database	64
Table	64
Column	64
Index	64
Constraint	64
Alias	256

Beginning with MySQL 4.1, identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions that are stored in `.frm` files and to identifiers stored in the grant tables in the `mysql` database. The sizes of the identifier string columns in the grant tables are measured in characters. You can use multi-byte characters without reducing the number of characters permitted for values stored in these columns, something not true prior to MySQL 4.1. The permissible Unicode characters are those in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted.

8.2.1 Identifier Qualifiers

MySQL permits names that consist of a single identifier or multiple identifiers. The components of a multiple-part name must be separated by period (“`.`”) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which the final identifier is interpreted.

In MySQL, you can refer to a table column using any of the following forms.

Column Reference	Meaning
`col_name`	The column *`col_name`* from whichever table used in the statement contains a column of that name.
`tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the default database.
`db_name.tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the database *`db_name`*. This syntax is unavailable before MySQL 3.22.

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write `` `my-table`.`my-column` ``, not `` `my-table.my-column` ``.

A reserved word that follows a period in a qualified name must be an identifier, so in that context it need not be quoted.

You need not specify a *`tbl_name`* or *`db_name.tbl_name`* prefix for a column reference in a statement unless the reference would be ambiguous. Suppose that tables `t1` and `t2` each contain a column `c`, and you retrieve `c` in a `SELECT` statement that uses both `t1` and `t2`. In this case, `c` is ambiguous because it is not unique among the tables used in the statement. You must qualify it with a table name as `t1.c` or `t2.c` to indicate which table you mean. Similarly, to retrieve from a table `t` in database `db1` and from a table `t` in database `db2` in the same statement, you must refer to columns in those tables as `db1.t.`*`col_name`* and `db2.t.`*`col_name`*.

The syntax `.`*`tbl_name`* means the table *`tbl_name`* in the default database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a “`.`” character.

8.2.2 Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Consequently, the case sensitivity of the underlying operating system plays a part in the case sensitivity of database and table names. This means database and table names are not case sensitive in Windows, and case sensitive in most varieties of Unix. One notable exception is Mac OS X, which is Unix-based but uses a default file system type (HFS+) that is not case sensitive. However, Mac OS X also supports UFS volumes, which are case sensitive just as on any Unix. See Section 1.9.4, "MySQL Extensions to Standard SQL". The `lower_case_table_names` [418] system variable also affects how the server handles identifier case sensitivity, as described later in this section.

Note

Although database and table names are not case sensitive on some platforms, you should not refer to a given database or table using different cases within the same statement. The following statement would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column and index names are not case sensitive on any platform, nor are column aliases.

Table aliases are case sensitive before MySQL 4.1.1. The following query would not work because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
    -> WHERE a.col_name = 1 OR A.col_name = 2;
```

If you have trouble remembering the permissible lettercase for database and table names, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` [418] system variable, which you can set when starting `mysqld`. `lower_case_table_names` [418] can take the values shown in the following table. On Unix, the default value of `lower_case_table_names` [418] is 0. On Windows, the default value is 1. On Mac OS X, the default is 1 before MySQL 4.0.18 and 2 as of 4.0.18.

Value	Meaning
`0`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement. Name comparisons are case sensitive. You should *not* set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or Mac OS X). If you force this variable to 0 with `--lower-case-table-names=0` [418] on a case-insensitive file system and access `MyISAM` tablenames using different lettercases, this may lead to index corruption.
`1`	Table names are stored in lowercase on disk and name comparisons are not case sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names as of MySQL 4.0.2, and to table aliases as of 4.1.1.
`2`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. This works *only* on file systems that are not case sensitive! `InnoDB` table names are stored in lowercase, as for `lower_case_table_names=1`. Setting `lower_case_table_names` [418] to `2` can be done as of MySQL 4.0.18.

If you are using MySQL on only one platform, you do not normally have to change the `lower_case_table_names` [418] variable from its default value. However, you may encounter difficulties if you want to transfer tables between platforms that differ in file system case sensitivity. For example, on Unix, you can have two different tables named `my_table` and `MY_TABLE`, but on Windows those names are considered identical. To avoid data transfer problems arising from lettercase of database or table names, you have two options:

- Use `lower_case_table_names=1` on all systems. The main disadvantage with this is that when you use `SHOW TABLES` or `SHOW DATABASES`, you do not see the names in their original lettercase.
- Use `lower_case_table_names=0` on Unix and `lower_case_table_names=2` on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

  Exception: If you are using `InnoDB` tables and you are trying to avoid these data transfer problems, you should set `lower_case_table_names` [418] to 1 on all platforms to force names to be converted to lowercase.

If you plan to set the `lower_case_table_names` [418] system variable to 1 on Unix, you must first convert your old database and table names to lowercase before stopping `mysqld` and restarting it with the new variable setting.

8.2.3 Function Name Parsing and Resolution

MySQL 4.1 supports built-in (native) functions and user-defined functions (UDFs). This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines which function to use in cases when functions of different types exist with a given name.

Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the `IGNORE_SPACE` [458] SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a nonexpression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to `count` is a function call, whereas the second reference is a table name:

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in nonexpression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in nonexpression context:

- To use the name as a function call in an expression, there must be no whitespace between the name and the following "`(`" parenthesis character.

- Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. `COUNT` is one such name. The exact list of function names for which following whitespace determines their interpretation are those listed in the `sql_functions[]` array of the `sql/lex.h` source file. Before MySQL 5.1, they are rather numerous (about 200), so you may find it easiest to treat the no-whitespace requirement as applying to all function calls. In MySQL 5.1, parser improvements reduce to about 30 the number of affected function names.

For functions not listed in the `sql_functions[]`) array, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. `ASCII` is one such name. However, for these nonaffected function names, interpretation may vary in expression context: *`func_name ()`* is interpreted as a built-in function if there is one with the given name; if not, *`func_name ()`* is interpreted as a user-defined function if one exists with that name.

The `IGNORE_SPACE` [458] SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

- With `IGNORE_SPACE` [458] disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in nonexpression context:

```
mysql> CREATE TABLE count(i INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'count(i INT)'
```

  To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

```
CREATE TABLE count (i INT);
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

- With `IGNORE_SPACE` [458] enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

```
SELECT COUNT(*) FROM mytable;
SELECT COUNT (*) FROM mytable;
```

  However, enabling `IGNORE_SPACE` [458] also has the side effect that the parser treats the affected function names as reserved words (see Section 8.3, “Reserved Words”). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in nonexpression context unless it is quoted. For example, with `IGNORE_SPACE` [458] enabled, both of the following statements fail with a syntax error because the parser interprets `count` as a reserved word:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

  To use the function name in nonexpression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

To enable the IGNORE_SPACE [458] SQL mode, use this statement:

```
SET sql_mode = 'IGNORE_SPACE';
```

IGNORE_SPACE [458] is also enabled by certain other composite modes such as ANSI [460] that include it in their value:

```
SET sql_mode = 'ANSI';
```

Check Section 5.1.6, “Server SQL Modes”, to see which composite modes enable IGNORE_SPACE [458].

To minimize the dependency of SQL code on the IGNORE_SPACE [458] setting, use these guidelines:

- Avoid creating UDFs that have the same name as a built-in function.
- Avoid using function names in nonexpression context. For example, these statements use count (one of the affected function names affected by IGNORE_SPACE [458]), so they fail with or without whitespace following the name if IGNORE_SPACE [458] is enabled:

  ```
  CREATE TABLE count(i INT);
  CREATE TABLE count (i INT);
  ```

  If you must use a function name in nonexpression context, write it as a quoted identifier:

  ```
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

Function Name Resolution

The server resolves references to function names for function creation and invocation as follows: A UDF can be created with the same name as a built-in function but the UDF cannot be invoked because the parser resolves invocations of the function to refer to the built-in function. For example, if you create a UDF named ABS, references to ABS() [817] invoke the built-in function.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions. If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use DROP FUNCTION to drop the UDF, and then use CREATE FUNCTION to re-create the UDF with a different nonconflicting name.

8.3 Reserved Words

Certain words such as SELECT, DELETE, or BIGINT are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Reserved words are permitted as identifiers if you quote them as described in Section 8.2, “Database, Table, Index, Column, and Alias Names”:

```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'

mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, `COUNT` is acceptable as a column name. However, by default, no whitespace is permitted in function invocations between the function name and the following "`(`" character. This requirement enables the parser to distinguish whether the name is used in a function call or in nonfunction context. For further detail on recognition of function names, see Section 8.2.3, "Function Name Parsing and Resolution".

The words in the following table are explicitly reserved in MySQL 4.1. At some point, you might upgrade to a higher version, so it is a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the words in the table are forbidden by standard SQL as column or table names (for example, `GROUP`). A few are reserved because MySQL needs them and uses a `yacc` parser. A reserved word can be used as an identifier if you quote it.

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	BEFORE	BETWEEN
BIGINT	BINARY	BLOB
BOTH	BY	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	COLUMNS	CONSTRAINT
CONVERT	CREATE	CROSS
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP
CURRENT_USER	DATABASE	DATABASES
DAY_HOUR	DAY_MICROSECOND	DAY_MINUTE
DAY_SECOND	DEC	DECIMAL
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DISTINCT
DISTINCTROW	DIV	DOUBLE
DROP	DUAL	ELSE
ENCLOSED	ESCAPED	EXISTS
EXPLAIN	FALSE	FIELDS
FLOAT	FLOAT4	FLOAT8
FOR	FORCE	FOREIGN
FROM	FULLTEXT	GRANT
GROUP	HAVING	HIGH_PRIORITY
HOUR_MICROSECOND	HOUR_MINUTE	HOUR_SECOND
IF	IGNORE	IN
INDEX	INFILE	INNER

INSERT	INT	INT1
INT2	INT3	INT4
INT8	INTEGER	INTERVAL
INTO	IS	JOIN
KEY	KEYS	KILL
LEADING	LEFT	LIKE
LIMIT	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOW_PRIORITY	MATCH	MEDIUMBLOB
MEDIUMINT	MEDIUMTEXT	MIDDLEINT
MINUTE_MICROSECOND	MINUTE_SECOND	MOD
NATURAL	NOT	NO_WRITE_TO_BINLOG
NULL	NUMERIC	ON
OPTIMIZE	OPTION	OPTIONALLY
OR	ORDER	OUTER
OUTFILE	PRECISION	PRIMARY
PRIVILEGES	PROCEDURE	PURGE
READ	REAL	REFERENCES
REGEXP	RENAME	REPLACE
REQUIRE	RESTRICT	REVOKE
RIGHT	RLIKE	SECOND_MICROSECOND
SELECT	SEPARATOR	SET
SHOW	SMALLINT	SONAME
SPATIAL	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SSL	STARTING
STRAIGHT_JOIN	TABLE	TABLES
TERMINATED	THEN	TINYBLOB
TINYINT	TINYTEXT	TO
TRAILING	TRUE	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	USAGE	USE
USING	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VALUES	VARBINARY
VARCHAR	VARCHARACTER	VARYING
WHEN	WHERE	WITH
WRITE	XOR	YEAR_MONTH
ZEROFILL		

The following are new reserved words in MySQL 4.0:

CHECK	FORCE	LOCALTIME
LOCALTIMESTAMP	REQUIRE	SQL_CALC_FOUND_ROWS
SSL	XOR	

The following are new reserved words in MySQL 4.1:

BEFORE	COLLATE	CONVERT
CURRENT_USER	DAY_MICROSECOND	DIV
DUAL	FALSE	HOUR_MICROSECOND
MINUTE_MICROSECOND	MOD	NO_WRITE_TO_BINLOG
SECOND_MICROSECOND	SEPARATOR	SPATIAL
TRUE	UTC_DATE	UTC_TIME
UTC_TIMESTAMP	VARCHARACTER	

MySQL permits some keywords to be used as unquoted identifiers because many people previously used them. Examples are those in the following list:

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

8.4 User-Defined Variables

MySQL supports user variables as of version 3.23.6. You can store a value in a user-defined variable in one statement and then refer to it later in another statement. This enables you to pass values from one statement to another. *User-defined variables are connection-specific*. That is, a user variable defined by one client cannot be seen or used by other clients. All variables for a given client connection are automatically freed when that client exits.

User variables are written as `@var_name`, where the variable name *`var_name`* consists of alphanumeric characters from the current character set, “.”, “_”, and “$”. A user variable name can contain other characters if you quote it as a string or identifier (for example, `@'my-var'`, `@"my-var"`, or `` @`my-var` ``). The default character set is `latin1` (cp1252 West European). This can be changed with the `--default-character-set` [385] option to `mysqld`. See Section 9.6, “Character Set Configuration”.

User variable names are not case sensitive in MySQL 5.0 and up, but are case sensitive before MySQL 5.0.

One way to set a user-defined variable is by issuing a `SET` statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For SET, either = [789] or := [788] can be used as the assignment operator.

You can also assign a value to a user variable in statements other than SET. In this case, the assignment operator must be := [788] and not = [789] because the latter is treated as the comparison operator = [782] in non-SET statements:

```
mysql> SET @t1=1, @t2=2, @t3:=4;
mysql> SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
+------+------+------+--------------------+
| @t1  | @t2  | @t3  | @t4 := @t1+@t2+@t3 |
+------+------+------+--------------------+
|    1 |    2 |    4 |                  7 |
+------+------+------+--------------------+
```

User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or nonbinary string, or NULL value. Assignment of decimal and real values does not preserve the precision or scale of the value. A value of a type other than one of the permissible types is converted to a permissible type. For example, a value having a temporal or spatial data type is converted to a binary string.

Beginning with MySQL 4.1.1, if a user variable is assigned a nonbinary (character) string value, it has the same character set and collation as the string. The coercibility of user variables is “implicit” as of MySQL 4.1.11 and 5.0.3. (This is the same coercibility as table column values.)

If the value of a user variable is selected in a result set, it is returned to the client as a string.

If you refer to a variable that has not been initialized, it has a value of NULL and a type of string.

User variables may be used in most contexts where expressions are permitted. This does not currently include contexts that explicitly require a literal value, such as in the LIMIT clause of a SELECT statement, or the IGNORE *N* LINES clause of a LOAD DATA statement.

As a general rule, you should never assign a value to a user variable and read the value within the same statement. You might get the results you expect, but this is not guaranteed. The order of evaluation for expressions involving user variables is undefined and may change based on the elements contained within a given statement. In SELECT @a, @a:=@a+1, ..., you might think that MySQL will evaluate @a first and then do an assignment second. However, changing the statement (for example, by adding a GROUP BY, HAVING, or ORDER BY clause) may cause MySQL to select an execution plan with a different order of evaluation.

Another issue with assigning a value to a variable and reading the value within the same statement is that the default result type of a variable is based on its type at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

For this SELECT statement, MySQL reports to the client that column one is a string and converts all accesses of @a to strings, even though @a is set to a number for the second row. After the SELECT statement executes, @a is regarded as a number for the next statement.

To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to 0, 0.0, or '' to define its type before you use it.

In a SELECT statement, each select expression is evaluated only when sent to the client. This means that in a HAVING, GROUP BY, or ORDER BY clause, referring to a variable that is assigned a value in the select expression list does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to b in the HAVING clause refers to an alias for an expression in the select list that uses @aa. This does not work as expected: @aa contains the value of id from the previous selected row, not from the current row.

User variables are intended to provide data values. They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected, or as a reserved word such as SELECT. This is true even if the variable is quoted, as shown in the following example:

```
mysql> SELECT c1 FROM t;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| c1   |
+------+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| `c1` |
+------+
1 row in set (0.00 sec)
```

An exception to this principle that user variables cannot be used to provide identifiers is that if you are constructing a string for use as a prepared statement to be executed later. In this case, user variables can be used to provide any part of the statement. The following example illustrates how this can be done:

```
mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
```

```
Query OK, 0 rows affected (0.04 sec)
Statement prepared

mysql> EXECUTE stmt;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

See Section 12.6, “SQL Syntax for Prepared Statements”, for more information.

PREPARE is available as of MySQL 4.1. Before MySQL 4.1, you must assemble a string for the query in application code, as shown here using PHP 5:

```
<?php
  $mysqli = new mysqli("localhost", "user", "pass", "test");

  if( mysqli_connect_errno() )
    die("Connection failed: %s\n", mysqli_connect_error());

  $col = "c1";

  $query = "SELECT $col FROM t";

  $result = $mysqli->query($query);

  while($row = $result->fetch_assoc())
  {
    echo "<p>" . $row["$col"] . "</p>\n";
  }

  $result->close();

  $mysqli->close();
?>
```

Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”.

8.5 Expression Syntax

The following rules define expression syntax in MySQL. The grammar shown here is based on that given in the `sql/sql_yacc.yy` file of MySQL source distributions. See the notes after the grammar for additional information about some of the terms. Operator precedence is given in Section 11.3.1, “Operator Precedence”.

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary
```

```
boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
  | boolean_primary comparison_operator {ALL | ANY} (subquery)
  | predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
    bit_expr [NOT] IN (subquery)
  | bit_expr [NOT] IN (expr [, expr] ...)
  | bit_expr [NOT] BETWEEN bit_expr AND predicate
  | bit_expr SOUNDS LIKE bit_expr
  | bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
  | bit_expr [NOT] REGEXP bit_expr
  | bit_expr

bit_expr:
    bit_expr | bit_expr
  | bit_expr & bit_expr
  | bit_expr << bit_expr
  | bit_expr >> bit_expr
  | bit_expr + bit_expr
  | bit_expr - bit_expr
  | bit_expr * bit_expr
  | bit_expr / bit_expr
  | bit_expr DIV bit_expr
  | bit_expr MOD bit_expr
  | bit_expr % bit_expr
  | bit_expr ^ bit_expr
  | bit_expr + interval_expr
  | bit_expr - interval_expr
  | simple_expr

simple_expr:
    literal
  | identifier
  | function_call
  | simple_expr COLLATE collation_name
  | param_marker
  | variable
  | simple_expr || simple_expr
  | + simple_expr
  | - simple_expr
  | ~ simple_expr
  | ! simple_expr
  | BINARY simple_expr
  | (expr [, expr] ...)
  | ROW (expr, expr [, expr] ...)
  | (subquery)
  | EXISTS (subquery)
  | {identifier expr}
  | match_expr
  | case_expr
  | interval_expr
```

Notes:

For literal value syntax, see Section 8.1, “Literal Values”.

For identifier syntax, see Section 8.2, “Database, Table, Index, Column, and Alias Names”.

Variables can be user variables, system variables, or stored program local variables or parameters:

- User variables: Section 8.4, “User-Defined Variables”

- System variables: Section 5.1.4, “Using System Variables”
- Local variables: Local Variable DECLARE Syntax
- Parameters: CREATE PROCEDURE and CREATE FUNCTION Syntax

param_marker is '?' as used in prepared statements for placeholders. See Section 12.6.1, “PREPARE Syntax”.

(*subquery*) indicates a subquery that returns a single value; that is, a scalar subquery. See Section 12.2.8.1, “The Subquery as Scalar Operand”.

{*identifier expr*} is ODBC escape syntax and is accepted for ODBC compatibility. The value is *expr*. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

match_expr indicates a MATCH [845] expression. See Section 11.9, “Full-Text Search Functions”.

case_expr indicates a CASE [790] expression. See Section 11.4, “Control Flow Functions”.

interval_expr represents a time interval. The syntax is INTERVAL *expr unit*, where *unit* is a specifier such as HOUR, DAY, or WEEK. For the full list of *unit* specifiers, see the description of the DATE_ADD() [829] function in Section 11.7, “Date and Time Functions”.

The meaning of the || [787] operator depends on the SQL mode. By default, || [787] is a logical OR [787] operator. With PIPES_AS_CONCAT [460] enabled, || [787] is string concatenation, with a precedence between ^ [863] and the unary operators. See Section 5.1.6, “Server SQL Modes”.

8.6 Comment Syntax

MySQL Server supports three comment styles:

- From a “#” character to the end of the line.
- From a “-- ” sequence to the end of the line. This style is supported as of MySQL 3.23.3. In MySQL, the “-- ” (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in Section 1.9.5.8, “'--' as the Start of a Comment”.
- From a /* sequence to the following */ sequence, as in the C programming language. This syntax enables a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

Nested comments are not supported.

MySQL Server supports some variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the "`!`" character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The comment syntax just described applies to how the `mysqld` server parses SQL statements. The `mysql` client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.) However, there are some limitations on the way that `mysql` parses `/* ... */` comments:

- A semicolon within the comment is taken to indicate the end of the current SQL statement and anything following it to indicate the beginning of the next statement. This problem was fixed in MySQL 4.0.13.
- A single quote, double quote, or backtick character is taken to indicate the beginning of a quoted string or identifier, even within a comment. If the quote is not matched by a second quote within the comment, the parser doesn't realize the comment has ended. If you are running `mysql` interactively, you can tell that it has gotten confused like this because the prompt changes from `mysql>` to `'>`, `">`, or `` `> ``. This problem was fixed in MySQL 4.1.1.
- The use of short-form commands such as `\C` within multi-line `/* ... */` comments is not supported.

Comments in this format, `/*!12345 ... */`, are not stored on the server. If this format is used to comment stored routines, the comments will not be retained on the server.

For affected versions of MySQL, these limitations apply both when you run `mysql` interactively and when you put commands in a file and use `mysql` in batch mode to process the file with `mysql < file_name`.

Chapter 9 Internationalization and Localization

Table of Contents

This chapter covers issues of internationalization (MySQL's capabilities for adapting to local use) and localization (selecting particular local conventions):

- MySQL support for character sets in SQL statements.
- How to configure the server to support different character sets.
- Selecting the language for error messages.
- How to set the server's time zone and enable per-connection time zone support.
- Selecting the locale for day and month names.

9.1 Character Set Support

Improved support for character set handling was added to MySQL in version 4.1. This support enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. You can specify character sets at the server, database, table, and column level. MySQL supports the use of character sets for the `MyISAM`, `MEMORY`, and (as of MySQL 4.1.2) `InnoDB` storage engines. The `ISAM`

storage engine does not include character set support; there are no plans to change this, because `ISAM` is deprecated.

Note

The `NDBCLUSTER` storage engine in MySQL 4.1 (available beginning with MySQL 4.1.3-Max) provides limited character set and collation support; see Section 15.1.4, “Known Limitations of MySQL Cluster”.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment
- Syntax for specifying character sets and collations
- Affected functions and operations
- Unicode support
- The character sets and collations that are available, with notes

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about configuring character sets for application use and character set-related issues in client/server communication, see Section 9.1.5, “Configuring the Character Set and Collation for Applications”, and Section 9.1.4, “Connection Character Sets and Collations”.

9.1.1 Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: “A”, “B”, “a”, “b”. We give each letter a number: “A” = 0, “B” = 1, “a” = 2, “b” = 3. The letter “A” is a symbol, the number 0 is the **encoding** for “A”, and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, “A” and “B”. The simplest way to do this is to look at the encodings: 0 for “A” and 1 for “B”. Because 0 is less than 1, we say “A” is less than “B”. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters “a” and “b” as equivalent to “A” and “B”; (2) then compare the encodings. We call this a *case-insensitive* collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just “A” and “B” but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether

to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German “Ö”), and for multiple-character mappings (such as the rule that “Ö” = “OE” in one of the two German collations).

MySQL 4.1 can do these things for you:

- Store strings using a variety of character sets
- Compare strings using a variety of collations
- Mix strings with different character sets or collations in the same server, the same database, or even the same table
- Enable specification of character set and collation at any level

In these respects, not only is MySQL 4.1 far more flexible than MySQL 4.0, it also is far ahead of most other database management systems. However, to use these features effectively, you need to know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

9.1.2 Character Sets and Collations in MySQL

The MySQL server can support multiple character sets. To list the available character sets, use the `SHOW CHARACTER SET` statement. A partial listing follows. For more complete information, see Section 9.1.12, “Character Sets and Collations That MySQL Supports”.

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
...
```

Any given character set always has at least one collation. It may have several collations. To list the collations for a character set, use the `SHOW COLLATION` statement. For example, to see the collations for the `latin1` (cp1252 West European) character set, use this statement to find those collation names that begin with `latin1`:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
```

```
| Collation           | Charset | Id | Default | Compiled | Sortlen |
+---------------------+---------+----+---------+----------+---------+
| latin1_german1_ci   | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci   | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci    | latin1  | 15 |         |          |       0 |
| latin1_german2_ci   | latin1  | 31 |         | Yes      |       2 |
| latin1_bin          | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci   | latin1  | 48 |         |          |       0 |
| latin1_general_cs   | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci   | latin1  | 94 |         |          |       0 |
+---------------------+---------+----+---------+----------+---------+
```

The `latin1` collations have the following meanings.

Collation	Meaning
`latin1_german1_ci`	German DIN-1
`latin1_swedish_ci`	Swedish/Finnish
`latin1_danish_ci`	Danish/Norwegian
`latin1_german2_ci`	German DIN-2
`latin1_bin`	Binary according to `latin1` encoding
`latin1_general_ci`	Multilingual (Western European)
`latin1_general_cs`	Multilingual (ISO Western European), case sensitive
`latin1_spanish_ci`	Modern Spanish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has one collation that is the *default collation*. For example, the default collation for `latin1` is `latin1_swedish_ci`. The output for `SHOW CHARACTER SET` indicates which collation is the default for each displayed character set.
- There is a convention for collation names: They start with the name of the character set with which they are associated, they usually include a language name, and they end with `_ci` (case insensitive), `_cs` (case sensitive), or `_bin` (binary).

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

9.1.3 Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The description in the following sections may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

`CHARACTER SET` is used in clauses that specify a character set. `CHARSET` can be used as a synonym for `CHARACTER SET`.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character

set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see Section 9.1.4, “Connection Character Sets and Collations”.

9.1.3.1 Server Character Set and Collation

MySQL Server has a server character set and a server collation. These can be set at server startup on the command line or in an option file and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start `mysqld`. You can use `--character-set-server` [385] for the character set. Along with it, you can add `--collation-server` [385] for the collation. If you don't specify a character set, that is the same as saying `--character-set-server=latin1` [385]. If you specify only a character set (for example, `latin1`) but not a collation, that is the same as saying `--character-set-server=latin1` [385] `--collation-server=latin1_swedish_ci` [385] because `latin1_swedish_ci` is the default collation for `latin1`. Therefore, the following three commands all have the same effect:

```
shell> mysqld
shell> mysqld --character-set-server=latin1
shell> mysqld --character-set-server=latin1 \
          --collation-server=latin1_swedish_ci
```

One way to change the settings is by recompiling. If you want to change the default server character set and collation when building from sources, use: `--with-charset` [97] and `--with-collation` [97] as arguments for `configure`. For example:

```
shell> ./configure --with-charset=latin1
```

Or:

```
shell> ./configure --with-charset=latin1 \
          --with-collation=latin1_german1_ci
```

Both `mysqld` and `configure` verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The server character set and collation are used as default values if the database character set and collation are not specified in `CREATE DATABASE` statements. They have no other purpose.

The current server character set and collation can be determined from the values of the `character_set_server` [408] and `collation_server` [409] system variables. These variables can be changed at runtime.

9.1.3.2 Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
```

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

All database options are stored in a text file named `db.opt` that can be found in the database directory.

The `CHARACTER SET` and `COLLATE` clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.
- If `CHARACTER SET X` is specified without `COLLATE`, character set *X* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *Y* and collation *Y* are used.
- Otherwise, the server character set and server collation are used.

The database character set and collation are used as default values for table definitions if the table character set and collation are not specified in `CREATE TABLE` statements. They have no other purpose.

The character set and collation for the default database can be determined from the values of the `character_set_database` [408] and `collation_database` [409] system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, `character_set_server` [408] and `collation_server` [409].

9.1.3.3 Table Character Set and Collation

Every table has a table character set and a table collation. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]]

ALTER TABLE tbl_name
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.
- If `CHARACTER SET X` is specified without `COLLATE`, character set *X* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *Y* and collation *Y* are used.
- Otherwise, the database character set and collation are used.

The table character set and collation are used as default values for column definitions if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

9.1.3.4 Column Character Set and Collation

Every “character” column (that is, a column of type `CHAR`, `VARCHAR`, or `TEXT`) has a column character set and a column collation. Column definition syntax for `CREATE TABLE` and `ALTER TABLE` has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

These clauses can also be used for `ENUM` and `SET` columns:

```
col_name {ENUM | SET} (val_list)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Examples:

```
CREATE TABLE t1
(
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_german1_ci
);

ALTER TABLE t1 MODIFY
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_swedish_ci;
```

MySQL chooses the column character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.

  ```
  CREATE TABLE t1
  (
      col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci
  ) CHARACTER SET latin1 COLLATE latin1_bin;
  ```

  The character set and collation are specified for the column, so they are used. The column has character set `utf8` and collation `utf8_unicode_ci`.

- If `CHARACTER SET X` is specified without `COLLATE`, character set *X* and its default collation are used.

```
CREATE TABLE t1
(
    col1 CHAR(10) CHARACTER SET utf8
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The character set is specified for the column, but the collation is not. The column has character set `utf8` and the default collation for `utf8`, which is `utf8_general_ci`. To see the default collation for each character set, use the `SHOW COLLATION` statement.

- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *`Y`* and collation *`Y`* are used.

```
CREATE TABLE t1
(
    col1 CHAR(10) COLLATE utf8_polish_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The collation is specified for the column, but the character set is not. The column has collation `utf8_polish_ci` and the character set is the one associated with the collation, which is `utf8`.

- Otherwise, the table character set and collation are used.

```
CREATE TABLE t1
(
    col1 CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_bin;
```

Neither the character set nor collation are specified for the column, so the table defaults are used. The column has character set `latin1` and collation `latin1_bin`.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

If you use `ALTER TABLE` to convert a column from one character set to another, MySQL attempts to map the data values, but if the character sets are incompatible, there may be data loss.

9.1.3.5 Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

A character string literal may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For the simple statement `SELECT 'string'`, the string has the character set and collation defined by the `character_set_connection` [408] and `collation_connection` [409] system variables.

The *`_charset_name`* expression is formally called an *introducer*. It tells the parser, “the string that is about to follow uses character set *`X`*.” Because this has confused people in the past, we emphasize that

an introducer does not change the string to the introducer character set like `CONVERT()` [859] would do. It does not change the string's value, although padding may occur. The introducer is just a signal. An introducer is also legal before standard hex literal and numeric hex literal notation (`x'literal'` and `0xnnnn`).

Examples:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
```

MySQL determines a literal's character set and collation in the following manner:

- If both `_X` and `COLLATE Y` are specified, character set `X` and collation `Y` are used.
- If `_X` is specified but `COLLATE` is not specified, character set `X` and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.
- Otherwise, the character set and collation given by the `character_set_connection` [408] and `collation_connection` [409] system variables are used.

Examples:

- A string with `latin1` character set and `latin1_german1_ci` collation:

  ```
  SELECT _latin1'Müller' COLLATE latin1_german1_ci;
  ```

- A string with `latin1` character set and its default collation (that is, `latin1_swedish_ci`):

  ```
  SELECT _latin1'Müller';
  ```

- A string with the connection default character set and collation:

  ```
  SELECT 'Müller';
  ```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

An introducer indicates the character set for the following string, but does not change now how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection` [408].

The following examples show that escape processing occurs using `character_set_connection` [408] even in the presence of an introducer. The examples use `SET NAMES` (which changes `character_set_connection` [408], as discussed in Section 9.1.4, “Connection Character Sets and Collations”), and display the resulting strings using the `HEX()` [796] function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+------------+-----------------+
| HEX('à\n') | HEX(_sjis'à\n') |
```

```
+------------+-----------------+
| E00A       | E00A            |
+------------+-----------------+
1 row in set (0.00 sec)
```

Here, “à” (hex value E0) is followed by “\n”, the escape sequence for newline. The escape sequence is interpreted using the character_set_connection [408] value of latin1 to produce a literal newline (hex value 0A). This happens even for the second string. That is, the introducer of _sjis does not affect the parser's escape processing.

Example 2:

```
mysql> SET NAMES sjis;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+------------+-------------------+
| HEX('à\n') | HEX(_latin1'à\n') |
+------------+-------------------+
| E05C6E     | E05C6E            |
+------------+-------------------+
1 row in set (0.04 sec)
```

Here, character_set_connection [408] is sjis, a character set in which the sequence of “à” followed by “\” (hex values 05 and 5C) is a valid multi-byte character. Hence, the first two bytes of the string are interpreted as a single sjis character, and the “\” is not interpreted as an escape character. The following “n” (hex value 6E) is not interpreted as part of an escape sequence. This is true even for the second string; the introducer of _latin1 does not affect escape processing.

9.1.3.6 National Character Set

Before MySQL 4.1, NCHAR and CHAR were synonymous. Standard SQL defines NCHAR or NATIONAL CHAR as a way to indicate that a CHAR column should use some predefined character set. MySQL 4.1 and up uses utf8 as that predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use N'*literal*' (or n'*literal*') to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

9.1.3.7 Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that is straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
    c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

9.1.3.8 Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

9.1.4 Connection Character Sets and Collations

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The server character set and collation can be determined from the values of the `character_set_server` [408] and `collation_server` [409] system variables.
- The character set and collation of the default database can be determined from the values of the `character_set_database` [408] and `collation_database` [409] system variables.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has connection-related character set and collation system variables.

Consider what a “connection” is: It is what you make when you connect to the server. The client sends SQL statements, such as queries, over the connection to the server. The server sends responses, such as result sets or error messages, over the connection back to the client. This leads to several questions about character set and collation handling for client connections, each of which can be answered in terms of system variables:

- What character set is the statement in when it leaves the client?

  The server takes the `character_set_client` [408] system variable to be the character set in which statements are sent by the client.

- What character set should the server translate a statement to after receiving it?

  For this, the server uses the `character_set_connection` [408] and `collation_connection` [409] system variables. It converts statements sent by the client from `character_set_client` [408] to `character_set_connection` [408] (except for string literals that have an introducer such as `_latin1` or `_utf8`). `collation_connection` [409] is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` [409] does not matter because columns have their own collation, which has a higher collation precedence.

- What character set should the server translate to before shipping result sets back to the client?

  The `character_set_results` [408] system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, and result metadata such as column names.

Clients can fine-tune the settings for these variables, or depend on the defaults (in which case, you can skip the rest of this section). If you do not use the defaults, you must change the character settings *for each connection to the server.*

There are two statements that affect the connection-related character set variables as a group:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

  `SET NAMES` indicates what character set the client will use to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server, “future incoming messages from this client are in character set `cp1251`.” It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement.)

  A `SET NAMES 'x'` statement is equivalent to these three statements:

```
SET character_set_client = x;
SET character_set_results = x;
SET character_set_connection = x;
```

Setting each of these character set variables also sets its corresponding collation variable to the default correlation for the character set. For example, setting `character_set_connection` [408] to *x* also sets `collation_connection` [409] to the default collation for *x*. It is not necessary to set that collation explicitly. To specify a particular collation for the character sets, use the optional `COLLATE` clause:

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

- `SET CHARACTER SET charset_name`

  `SET CHARACTER SET` is similar to `SET NAMES` but sets `character_set_connection` [408] and `collation_connection` [409] to `character_set_database` [408] and `collation_database` [409]. A `SET CHARACTER SET x` statement is equivalent to these three statements:

```
SET character_set_client = x;
SET character_set_results = x;
SET collation_connection = @@collation_database;
```

  Setting `collation_connection` [409] also sets `character_set_connection` [408] to the character set associated with the collation (equivalent to executing `SET character_set_connection = @@character_set_database`). It is not necessary to set `character_set_connection` [408] explicitly.

Note

`ucs2` cannot be used as a client character set, which means that it does not work for `SET NAMES` or `SET CHARACTER SET`.

The MySQL client programs `mysql`, `mysqladmin`, `mysqlcheck`, `mysqlimport`, and `mysqlshow` determine the default character set to use as follows:

- In the absence of other information, the programs use the compiled-in default character set, usually `latin1`.
- The programs support a `--default-character-set` [260] option, which enables users to specify the character set explicitly to override whatever default the client otherwise determines.

When a client connects to the server, it sends the name of the character set that it wants to use. The server uses the name to set the `character_set_client` [408], `character_set_results` [408], and `character_set_connection` [408] system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

With the `mysql` client, if you want to use a character set different from the default, you could explicitly execute `SET NAMES` every time you start up. However, to accomplish the same result more easily, you can add the `--default-character-set` [260] option setting to your `mysql` command line or in your option file. For example, the following option file setting changes the three connection-related character set variables set to `koi8r` each time you invoke `mysql`:

```
[mysql]
default-character-set=koi8r
```

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET latin1` before issuing the `SELECT` statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy if there are characters that are not in both character sets.

If you do not want the server to perform any conversion of result sets or error messages, set `character_set_results` [408] to `NULL` or `binary`:

```
SET character_set_results = NULL;
```

To see the values of the character set and collation system variables that apply to your connection, use these statements:

```
SHOW VARIABLES LIKE 'character_set%';
SHOW VARIABLES LIKE 'collation%';
```

You must also consider the environment within which your MySQL applications execute. See Section 9.1.5, “Configuring the Character Set and Collation for Applications”.

For more information about character sets and error messages, see Section 9.1.6, “Character Set for Error Messages”.

9.1.5 Configuring the Character Set and Collation for Applications

For applications that store data using the default MySQL character set and collation (`latin1`, `latin1_swedish_ci`), no special configuration should be needed. If applications require data storage using a different character set or collation, you can configure character set information several ways:

- Specify character settings per database. For example, applications that use one database might require `utf8`, whereas applications that use another database might require `sjis`.
- Specify character settings at server startup. This causes the server to use the given settings for all applications that do not make other arrangements.
- Specify character settings at configuration time, if you build MySQL from source. This causes the server to use the given settings for all applications, without having to specify them at server startup.

When different applications require different character settings, the per-database technique provides a good deal of flexibility. If most or all applications use the same character set, specifying character settings at server startup or configuration time may be most convenient.

For the per-database or server-startup techniques, the settings control the character set for data storage. Applications must also tell the server which character set to use for client/server communications, as described in the following instructions.

The examples shown here assume use of the `utf8` character set and `utf8_general_ci` collation.

Specify character settings per database. To create a database such that its tables will use a given default character set and collation for data storage, use a `CREATE DATABASE` statement like this:

```
CREATE DATABASE mydb
  DEFAULT CHARACTER SET utf8
```

```
  DEFAULT COLLATE utf8_general_ci;
```

Tables created in the database will use `utf8` and `utf8_general_ci` by default for any character columns.

Applications that use the database should also configure their connection to the server each time they connect. This can be done by executing a `SET NAMES 'utf8'` statement after connecting. The statement can be used regardless of connection method: The `mysql` client, PHP scripts, and so forth.

In some cases, it may be possible to configure the connection to use the desired character set some other way. For example, for connections made using `mysql`, you can specify the `--default-character-set=utf8` [260] command-line option to achieve the same effect as `SET NAMES 'utf8'`.

For more information about configuring client connections, see Section 9.1.4, “Connection Character Sets and Collations”.

Specify character settings at server startup. To select a character set and collation at server startup, use the `--character-set-server` [385] and `--collation-server` [385] options. For example, to specify the options in an option file, include these lines:

```
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
```

These settings apply server-wide and apply as the defaults for databases created by any application, and for tables created in those databases.

It is still necessary for applications to configure their connection using `SET NAMES` or equivalent after they connect, as described previously. You might be tempted to start the server with the `--init_connect="SET NAMES 'utf8'"` [414] option to cause `SET NAMES` to be executed automatically for each client that connects. However, this will yield inconsistent results because the `init_connect` [414] value is not executed for users who have the `SUPER` [493] privilege.

Specify character settings at MySQL configuration time. To select a character set and collation when you configure and build MySQL from source, use the `--with-charset` [97] and `--with-collation` [97] options:

```
shell> ./configure --with-charset=utf8 --with-collation=utf8_general_ci
```

The resulting server uses `utf8` and `utf8_general_ci` as the default for databases and tables and for client connections. It is unnecessary to use `--character-set-server` [385] and `--collation-server` [385] to specify those defaults at server startup. It is also unnecessary for applications to configure their connection using `SET NAMES` or equivalent after they connect to the server.

Regardless of how you configure the MySQL character set for application use, you must also consider the environment within which those applications execute. If you will send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file encoding is correct and so that the operating system handles it correctly. If you use the `mysql` client from within a terminal window, the window must be configured to use UTF-8 or characters may not display properly. For a script that executes in a Web environment, the script must handle character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages. For example, you can include this `<meta>` tag within your `<head>` element:

```
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

9.1.6 Character Set for Error Messages

This section describes how the server uses character sets for constructing error messages and returning them to clients. For information about the language of error messages (rather than the character set), see Section 9.3, “Setting the Error Message Language”.

In MySQL 4.1, the server constructs error messages and returns them to clients as follows:

- The message template has the character set associated with the error message language. For example, English, Korean, and Russian messages use `latin1`, `euckr`, and `koi8r`, respectively.
- Parameters in the message template are replaced with values that apply to a specific error occurrence. These parameters use their own character set. Identifiers such as table or column names use UTF-8. Data values retain their character set. For example, in the following duplicate-key message, `'xxx'` has the character set of the table column associated with key 1:

```
Duplicate entry 'xxx' for key1
```

The preceding method of error-message construction can result in messages that contain a mix of character sets unless all items involved contain only ASCII characters. This issue is resolved in MySQL 5.5, in which error messages are constructed internally within the server using UTF-8 and returned to the client in the character set specified by the `character_set_results` [408] system variable.

9.1.7 Collation Issues

The following sections discuss various aspects of character set collations.

9.1.7.1 Collation Names

MySQL collation names follow these rules:

- A name ending in `_ci` indicates a case-insensitive collation.
- A name ending in `_cs` indicates a case-sensitive collation.
- A name ending in `_bin` indicates a binary collation. Character comparisons are based on character binary code values.

9.1.7.2 Using `COLLATE` in SQL Statements

With the `COLLATE` clause, you can override whatever the default collation is for a comparison. `COLLATE` may be used in various parts of SQL statements. Here are some examples:

- With `ORDER BY`:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- With `AS`:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
```

```
ORDER BY k1;
```

- With `GROUP BY`:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With `DISTINCT`:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With `WHERE`:

```
    SELECT *
    FROM t1
    WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
    SELECT *
    FROM t1
    WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- With `HAVING`:

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

9.1.7.3 `COLLATE` Clause Precedence

The `COLLATE` clause has high precedence (higher than `||` [787]), so the following two expressions are equivalent:

```
x || y COLLATE z
x || (y COLLATE z)
```

9.1.7.4 Collations Must Be for the Right Character Set

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

In some cases, expressions that worked before MySQL 4.1 fail in early versions of MySQL 4.1 if you do not take character set and collation into account. For example, before 4.1, this statement works as is:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
+-----------------------------+
| SUBSTRING_INDEX(USER(),'@',1) |
+-----------------------------+
| root                        |
+-----------------------------+
```

The statement also works as is in MySQL 4.1 as of 4.1.8: In MySQL 4.1, user names are stored using the `utf8` character set (see Section 9.1.10, “UTF-8 for Metadata”). The literal string `'@'` has the server character set (`latin1` by default). Although the character sets are different, MySQL can coerce the `latin1` string to the character set (and collation) of `USER()` [876] without data loss. It does so, performs the substring operation, and returns a result that has a character set of `utf8`.

However, in versions of MySQL 4.1 before 4.1.8, the statement fails:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
ERROR 1267 (HY000): Illegal mix of collations
(utf8_general_ci,IMPLICIT) and (latin1_swedish_ci,COERCIBLE)
for operation 'substr_index'
```

This happens because the automatic character set conversion of `'@'` does not occur and the string operands have different character sets (and thus different collations):

```
mysql> SELECT COLLATION(USER()), COLLATION('@');
+-------------------+-------------------+
| COLLATION(USER()) | COLLATION('@')    |
+-------------------+-------------------+
| utf8_general_ci   | latin1_swedish_ci |
+-------------------+-------------------+
```

One way to deal with this is to upgrade to MySQL 4.1.8 or later. If that is not possible, you can tell MySQL to interpret the literal string as `utf8`:

```
mysql> SELECT SUBSTRING_INDEX(USER(),_utf8'@',1);
+-----------------------------------+
| SUBSTRING_INDEX(USER(),_utf8'@',1) |
+-----------------------------------+
| root                              |
+-----------------------------------+
```

Another way is to change the connection character set and collation to `utf8`. You can do that with `SET NAMES 'utf8'` or by setting the `character_set_connection` [408] and `collation_connection` [409] system variables directly.

9.1.7.5 Collation of Expressions

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column `x`:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, with multiple operands, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should the comparison use the collation of the column `x`, or of the string literal `'Y'`? Both `x` and `'Y'` have collations, so which collation takes precedence?

Standard SQL resolves such questions using what used to be called “coercibility” rules. MySQL assigns coercibility values as follows:

- An explicit `COLLATE` clause has a coercibility of 0. (Not coercible at all.)
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column has a coercibility of 2.
- A “system constant” (the string returned by functions such as `USER()` [876] or `VERSION()` [876]) has a coercibility of 3.
- The collation of a literal has a coercibility of 4.
- `NULL` or an expression that is derived from `NULL` has a coercibility of 5.

The preceding coercibility values are current as of MySQL 4.1.11. Before MySQL 4.1.11, there is no system constant or `NULL` coercibility. Functions such as `USER()` [876] have a coercibility of 2 rather than 3, and literals have a coercibility of 3 rather than 4.

MySQL uses coercibility values with the following rules to resolve ambiguities:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then:
  - If both sides are Unicode, or both sides are not Unicode, it is an error.
  - If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement does not return an error:

    ```
    SELECT CONCAT(utf8_column, latin1_column) FROM t1;
    ```

    It returns a result that has a character set of `utf8` and the same collation as `utf8_column`. Values of `latin1_column` are automatically converted to `utf8` before concatenating.
  - For an operation with operands from the same character set but that mix a `_bin` collation and a `_ci` or `_cs` collation, the `_bin` collation is used. This is similar to how operations that mix nonbinary and binary strings evaluate the operands as binary strings, except that it is for collations rather than data types.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

Examples:

Comparison	Collation Used
`column1 = 'A'`	Use collation of `column1`

Comparison	Collation Used
`column1 = 'A' COLLATE x`	Use collation of `'A' COLLATE x`
`column1 COLLATE x = 'A' COLLATE y`	Error

The `COERCIBILITY()` [871] function can be used to determine the coercibility of a string expression:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(VERSION());
        -> 3
mysql> SELECT COERCIBILITY('A');
        -> 4
```

See Section 11.13, “Information Functions”.

For implicit conversion of a numeric or temporal value to a string, such as occurs for the argument `1` in the expression `CONCAT(1, 'abc')` [795], the result is a binary string for which the character set and collation are `binary`. See Section 11.2, “Type Conversion in Expression Evaluation”.

9.1.7.6 The `_bin` and `binary` Collations

This section describes how `_bin` collations for nonbinary strings differ from the `binary` “collation” for binary strings.

Nonbinary strings (as stored in the `CHAR`, `VARCHAR`, and `TEXT` data types) have a character set and collation. A given character set can have several collations, each of which defines a particular sorting and comparison order for the characters in the set. One of these is the binary collation for the character set, indicated by a `_bin` suffix in the collation name. For example, `latin1` and `utf8` have binary collations named `latin1_bin` and `utf8_bin`.

Binary strings (as stored in the `BINARY`, `VARBINARY`, and `BLOB` data types) have no character set or collation in the sense that nonbinary strings do. (Applied to a binary string, the `CHARSET()` and `COLLATION()` functions both return a value of `binary`.) Binary strings are sequences of bytes and the numeric values of those bytes determine sort order.

The `_bin` collations differ from the `binary` collation in several respects.

The unit for sorting and comparison. Binary strings are sequences of bytes. Sorting and comparison is always based on numeric byte values. Nonbinary strings are sequences of characters, which might be multi-byte. Collations for nonbinary strings define an ordering of the character values for sorting and comparison. For the `_bin` collation, this ordering is based solely on binary code values of the characters (which is similar to ordering for binary strings except that a `_bin` collation must take into account that a character might contain multiple bytes). For other collations, character ordering might take additional factors such as lettercase into account.

Character set conversion. A nonbinary string has a character set and is converted to another character set in many cases, even when the string has a `_bin` collation:

- When assigning column values from another column that has a different character set:

```
UPDATE t1 SET utf8_bin_column=latin1_column;
INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
```

- When assigning column values for `INSERT` or `UPDATE` using a string literal:

```
SET NAMES latin1;
INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
```

- When sending results from the server to a client:

```
SET NAMES latin1;
SELECT utf8_bin_column FROM t2;
```

For binary string columns, no conversion occurs. For the preceding cases, the string value is copied byte-wise.

Lettercase conversion. Collations provide information about lettercase of characters, so characters in a nonbinary string can be converted from one lettercase to another, even for `_bin` collations that ignore lettercase for ordering:

```
mysql> SET NAMES latin1 COLLATE latin1_bin;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT LOWER('aA'), UPPER('zZ');
+-------------+-------------+
| LOWER('aA') | UPPER('zZ') |
+-------------+-------------+
| aa          | ZZ          |
+-------------+-------------+
1 row in set (0.13 sec)
```

The concept of lettercase does not apply to bytes in a binary string. To perform lettercase conversion, the string must be converted to a nonbinary string:

```
mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING latin1));
+-------------+-----------------------------------+
| LOWER('aA') | LOWER(CONVERT('aA' USING latin1)) |
+-------------+-----------------------------------+
| aA          | aa                                |
+-------------+-----------------------------------+
1 row in set (0.00 sec)
```

Trailing space handling in comparisons. Nonbinary strings have `PADSPACE` behavior for all collations, including `_bin` collations. Trailing spaces are insignificant in comparisons:

```
mysql> SET NAMES utf8 COLLATE utf8_bin;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          1 |
+------------+
1 row in set (0.00 sec)
```

For binary strings, all characters are significant in comparisons, including trailing spaces:

```
mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          0 |
+------------+
1 row in set (0.00 sec)
```

Trailing space handling for inserts and retrievals. `CHAR(N)` columns store nonbinary strings. Values shorter than *N* characters are extended with spaces on insertion. For retrieval, trailing spaces are removed.

`BINARY(N)` columns store binary strings. Values shorter than *N* bytes are extended with `0x00` bytes on insertion. For retrieval, nothing is removed; a value of the declared length is always returned.

```
mysql> CREATE TABLE t1 (
    ->   a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
    ->   b BINARY(10)
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t1 VALUES ('a','a');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(a), HEX(b) FROM t1;
+--------+----------------------+
| HEX(a) | HEX(b)               |
+--------+----------------------+
| 61     | 61000000000000000000 |
+--------+----------------------+
1 row in set (0.04 sec)
```

9.1.7.7 The BINARY Operator

The `BINARY` [859] operator casts the string following it to a binary string. This is an easy way to force a comparison to be done byte by byte rather than character by character. `BINARY` [859] also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
mysql> SELECT 'a' = 'a ';
        -> 1
mysql> SELECT BINARY 'a' = 'a ';
        -> 0
```

`BINARY str` is shorthand for `CAST(str AS BINARY)` [859].

The `BINARY` attribute in character column definitions has a different effect. A character column defined with the `BINARY` attribute is assigned the binary collation of the column character set. Every character set has a binary collation. For example, the binary collation for the `latin1` character set is `latin1_bin`, so if the table default character set is `latin1`, these two column definitions are equivalent:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin
```

The effect of `BINARY` as a column attribute differs from its effect prior to MySQL 4.1. Formerly, `BINARY` resulted in a column that was treated as a binary string. A binary string is a string of bytes that has no character set or collation, which differs from a nonbinary character string that has a binary collation. For both types of strings, comparisons are based on the numeric values of the string unit, but for nonbinary

strings the unit is the character and some character sets support multi-byte characters. Section 10.4.2, “The BINARY and VARBINARY Types”.

The use of CHARACTER SET binary in the definition of a CHAR, VARCHAR, or TEXT column causes the column to be treated as a binary data type. For example, the following pairs of definitions are equivalent:

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

9.1.7.8 Examples of the Effect of Collation

Example 1: Sorting German Umlauts

Suppose that column X in table T has these latin1 column values:

```
Muffler
Müller
MX Systems
MySQL
```

Suppose also that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

The following table shows the resulting order of the values if we use ORDER BY with different collations.

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call “U-umlaut.”

- The first column shows the result of the SELECT using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the SELECT using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the SELECT using the German DIN-2 rule, which says that U-umlaut sorts with UE.

Example 2: Searching for German Umlauts

Suppose that you have three tables that differ only by the character set and collation used:

```
mysql> SET NAMES utf8;
mysql> CREATE TABLE german1 (
    ->   c CHAR(10)
    -> ) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
    ->   c CHAR(10)
    -> ) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
    ->   c CHAR(10)
    -> ) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Each table contains two records:

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

Two of the above collations have an `A = Ä` equality, and one has no such equality (`latin1_german2_ci`). For that reason, you'll get these results in comparisons:

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bär  |
+------+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
```

This is not a bug but rather a consequence of the sorting properties of `latin1_german1_ci` and `utf8_unicode_ci` (the sorting shown is done according to the German DIN 5007 standard).

9.1.8 Operations Affected by Character Set Support

This section describes operations that take character set information into account as of MySQL 4.1.

9.1.8.1 Result Strings

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(X)` [804] returns a string whose character string and collation are the same as that of *X*. The same applies for `INSTR()` [797], `LCASE()` [797], `LOWER()` [798], `LTRIM()` [799], `MID()` [799], `REPEAT()` [800], `REPLACE()` [800], `REVERSE()` [800], `RIGHT()` [800], `RPAD()` [800], `RTRIM()` [801], `SOUNDEX()` [801], `SUBSTRING()` [802], `TRIM()` [803], `UCASE()` [803], and `UPPER()` [804].

Note: The REPLACE() [800] function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has no character set or collation. This can be checked by using the CHARSET() [870] and COLLATION() [871] functions, both of which return binary to indicate that their argument is a binary string:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+--------------------+----------------------+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+--------------------+----------------------+
| binary              | binary               |
+--------------------+----------------------+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit COLLATE *X* occurs, use *X*.
- If explicit COLLATE *X* and COLLATE *Y* occur, raise an error.
- Otherwise, if all collations are *X*, use *X*.
- Otherwise, the result has no collation.

For example, with CASE ... WHEN a THEN b WHEN b THEN c COLLATE *X* END, the resulting collation is *X*. The same applies for UNION, || [787], CONCAT() [795], ELT() [795], GREATEST() [784], IF() [790], and LEAST() [786].

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the character_set_connection [408] and collation_connection [409] system variables. This applies only to CAST() [859], CHAR() [794], CONV() [818], FORMAT() [796], HEX() [796], and SPACE() [801].

If you are uncertain about the character set or collation of the result returned by a string function, you can use the CHARSET() [870] or COLLATION() [871] function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+---------------+-----------------+-------------------+
| USER()        | CHARSET(USER()) | COLLATION(USER()) |
+---------------+-----------------+-------------------+
| test@localhost | utf8            | utf8_general_ci   |
+---------------+-----------------+-------------------+
```

9.1.8.2 CONVERT() and CAST()

CONVERT() [859] provides a way to convert data between different character sets. The syntax is:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
```

```
    SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` [859] is implemented according to the standard SQL specification.

You may also use `CAST()` [859] to convert a string to a different character set. The syntax is:

```
CAST(character_string AS character_data_type CHARACTER SET charset_name)
```

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

If you use `CAST()` [859] without specifying `CHARACTER SET`, the resulting character set and collation are defined by the `character_set_connection` [408] and `collation_connection` [409] system variables. If you use `CAST()` [859] with `CHARACTER SET X`, the resulting character set and collation are `X` and the default collation of `X`.

You may not use a `COLLATE` clause inside a `CONVERT()` [859] or `CAST()` [859] call, but you may use it outside. For example, `CAST(... COLLATE ...)` [859] is illegal, but `CAST(...) COLLATE ...` [859] is legal:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

9.1.8.3 `SHOW` Statements and `INFORMATION_SCHEMA`

Several `SHOW` statements are added or modified in MySQL 4.1 to provide additional character set information. `SHOW CHARACTER SET`, `SHOW COLLATION`, and `SHOW CREATE DATABASE` are new. `SHOW CREATE TABLE` and `SHOW COLUMNS` are modified. These statements are described here briefly. For more information, see Section 12.4.5, “`SHOW` Syntax”.

The `SHOW CHARACTER SET` statement shows all available character sets. It takes an optional `LIKE` [804] clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

The output from `SHOW COLLATION` includes all available character sets. It takes an optional `LIKE` [804] clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
```

```
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

`SHOW CREATE DATABASE` displays the `CREATE DATABASE` statement that creates a given database:

```
mysql> SHOW CREATE DATABASE test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
```

If no `COLLATE` clause is shown, the default collation for the character set applies.

`SHOW CREATE TABLE` is similar, but displays the `CREATE TABLE` statement to create a given table. The column definitions indicate any character set specifications, and the table options include character set information.

The `SHOW COLUMNS` statement displays the collations of a table's columns when invoked as `SHOW FULL COLUMNS`. Columns with `CHAR`, `VARCHAR`, or `TEXT` data types have collations. Numeric and other noncharacter types have no collation (indicated by `NULL` as the `Collation` value). For example:

```
mysql> SHOW FULL COLUMNS FROM person\G
*************************** 1. row ***************************
     Field: id
      Type: smallint(5) unsigned
 Collation: NULL
      Null: NO
       Key: PRI
   Default: NULL
     Extra: auto_increment
Privileges: select,insert,update,references
   Comment:
*************************** 2. row ***************************
     Field: name
      Type: char(60)
 Collation: latin1_swedish_ci
      Null: NO
       Key:
   Default:
     Extra:
Privileges: select,insert,update,references
   Comment:
```

The character set is not part of the display but is implied by the collation name.

9.1.9 Unicode Support

As of MySQL version 4.1, there are two new character sets for storing Unicode data:

- `ucs2`, the UCS-2 encoding of the Unicode character set using 16 bits per character
- `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character

These two character sets support the characters from the Basic Multilingual Plane (BMP) of Unicode Version 3.0. BMP characters have these characteristics:

- Their code values are between 0 and 65535 (or `U+0000` .. `U+FFFF`)

- They can be encoded with a fixed 16-bit word, as in `ucs2`
- They can be encoded with 8, 16, or 24 bits, as in `utf8`
- They are sufficient for almost all characters in major languages

The `ucs2` and `utf8` character sets do not support supplementary characters that lie outside the BMP.

A similar set of collations is available for each Unicode character set. For example, each has a Danish collation, the names of which are `ucs2_danish_ci` and `utf8_danish_ci`. All Unicode collations are listed at Section 9.1.12.1, “Unicode Character Sets”.

The MySQL implementation of UCS-2 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of values. Other database systems might use little-endian byte order or a BOM. In such cases, conversion of values will need to be performed when transferring data between those systems and MySQL.

MySQL uses no BOM for UTF-8 values.

Client applications that need to communicate with the server using Unicode should set the client character set accordingly; for example, by issuing a `SET NAMES 'utf8'` statement. `ucs2` cannot be used as a client character set, which means that it does not work for `SET NAMES` or `SET CHARACTER SET`. (See Section 9.1.4, “Connection Character Sets and Collations”.)

The following sections provide additional detail on the Unicode character sets in MySQL.

9.1.9.1 The `ucs2` Character Set (UCS-2 Unicode Encoding)

In UCS-2, every character is represented by a two-byte Unicode code with the most significant byte first. For example: `LATIN CAPITAL LETTER A` has the code `0x0041` and it is stored as a two-byte sequence: `0x00 0x41`. `CYRILLIC SMALL LETTER YERU` (Unicode `0x044B`) is stored as a two-byte sequence: `0x04 0x4B`. For Unicode characters and their codes, please refer to the Unicode Home Page.

In MySQL, the `ucs2` character set is a fixed-length 16-bit encoding for Unicode BMP characters.

9.1.9.2 The `utf8` Character Set (Three-Byte UTF-8 Unicode Encoding)

UTF-8 (Unicode Transformation Format with 8-bit units) is an alternative way to store Unicode data. It is implemented according to RFC 3629, which describes encoding sequences that take from one to four bytes. Currently, MySQL support for UTF-8 does not include four-byte sequences. (An older standard for UTF-8 encoding, RFC 2279, describes UTF-8 sequences that take from one to six bytes. RFC 3629 renders RFC 2279 obsolete; for this reason, sequences with five and six bytes are no longer used.)

The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a two-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use three-byte sequences.

Tip: To save space with UTF-8, use `VARCHAR` instead of `CHAR`. Otherwise, MySQL must reserve three bytes for each character in a `CHAR CHARACTER SET utf8` column because that is the maximum

possible length. For example, MySQL must reserve 30 bytes for a `CHAR(10) CHARACTER SET utf8` column.

9.1.10 UTF-8 for Metadata

Metadata is "the data about the data." Anything that *describes* the database—as opposed to being the *contents* of the database—is metadata. Thus column names, database names, user names, version names, and most of the string results from `SHOW` are metadata.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, `SHOW` wouldn't work properly because different rows in the same column would be in different character sets.
- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()` [876], `CURRENT_USER()` [872], `SESSION_USER()` [876], `SYSTEM_USER()` [876], `DATABASE()` [872], and `VERSION()` [876] functions have the UTF-8 character set by default.

The server sets the `character_set_system` [409] system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_system | utf8  |
+----------------------+-------+
```

Storage of metadata using Unicode does *not* mean that the server returns headers of columns and the results of `DESCRIBE` functions in the `character_set_system` [409] character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` [408] system variable, which has a default value of `latin1`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` [408] and other related system variables. (See Section 9.1.4, "Connection Character Sets and Collations".) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client perform the conversion, but this option is not available for many clients until late in the MySQL 4.x product cycle.

If `character_set_results` [408] is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system` [409]).

Beginning with MySQL 4.1.1, error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` [876] function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` [876] are automatically converted to `latin1` before the assignment.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. For more information about coercion of strings, see Section 9.1.7.5, “Collation of Expressions”.

9.1.11 Upgrading Character Sets from MySQL 4.0

What about upgrading from older versions of MySQL? MySQL 4.1 is almost upward compatible with MySQL 4.0 and earlier for the simple reason that almost all the features are new, so there is nothing in earlier versions to conflict with. However, there are some differences and a few things to be aware of.

It is important to note that the “MySQL 4.0 character set” contains both character set and collation information in one single entity. Beginning in MySQL 4.1, character sets and collations are separate entities. Though each collation corresponds to a particular character set, the two are not bundled together.

There is a special treatment of national character sets in MySQL 4.1. `NCHAR` is not the same as `CHAR`, and `N'...'` literals are not the same as `'...'` literals.

Finally, there is a different file format for storing information about character sets and collations. Make sure that you have reinstalled the `/share/mysql/charsets/` directory containing the new configuration files.

If you want to start `mysqld` from a 4.1.x distribution with data created by MySQL 4.0, you should start the server with the same character set and collation. In this case, you won't need to reindex your data.

There are two ways to do so:

```
shell> ./configure --with-charset=... --with-collation=...
shell> ./mysqld --default-character-set=... --default-collation=...
```

If you used `mysqld` with, for example, the MySQL 4.0 `danish` character set, you should use the `latin1` character set and the `latin1_danish_ci` collation:

```
shell> ./configure --with-charset=latin1 \
           --with-collation=latin1_danish_ci
shell> ./mysqld --default-character-set=latin1 \
           --default-collation=latin1_danish_ci
```

Use the table shown in Section 9.1.11.1, “4.0 Character Sets and Corresponding 4.1 Character Set/Collation Pairs”, to find old 4.0 character set names and their 4.1 character set/collation pair equivalents.

If you have non-`latin1` data stored in a 4.0 `latin1` table and want to convert the table column definitions to reflect the actual character set of the data, use the instructions in Section 9.1.11.2, “Converting 4.0 Character Columns to 4.1 Format”.

9.1.11.1 4.0 Character Sets and Corresponding 4.1 Character Set/Collation Pairs

ID	4.0 Character Set	4.1 Character Set	4.1 Collation
1	big5	big5	big5_chinese_ci
2	czech	latin2	latin2_czech_ci
3	dec8	dec8	dec8_swedish_ci
4	dos	cp850	cp850_general_ci
5	german1	latin1	latin1_german1_ci
6	hp8	hp8	hp8_english_ci
7	koi8_ru	koi8r	koi8r_general_ci
8	latin1	latin1	latin1_swedish_ci
9	latin2	latin2	latin2_general_ci
10	swe7	swe7	swe7_swedish_ci
11	usa7	ascii	ascii_general_ci
12	ujis	ujis	ujis_japanese_ci
13	sjis	sjis	sjis_japanese_ci
14	cp1251	cp1251	cp1251_bulgarian_ci
15	danish	latin1	latin1_danish_ci
16	hebrew	hebrew	hebrew_general_ci
17	win1251	(removed)	(removed)
18	tis620	tis620	tis620_thai_ci
19	euc_kr	euckr	euckr_korean_ci
20	estonia	latin7	latin7_estonian_ci
21	hungarian	latin2	latin2_hungarian_ci
22	koi8_ukr	koi8u	koi8u_ukrainian_ci
23	win1251ukr	cp1251	cp1251_ukrainian_ci
24	gb2312	gb2312	gb2312_chinese_ci
25	greek	greek	greek_general_ci
26	win1250	cp1250	cp1250_general_ci
27	croat	latin2	latin2_croatian_ci
28	gbk	gbk	gbk_chinese_ci
29	cp1257	cp1257	cp1257_lithuanian_ci
30	latin5	latin5	latin5_turkish_ci
31	latin1_de	latin1	latin1_german2_ci

9.1.11.2 Converting 4.0 Character Columns to 4.1 Format

Normally, the server runs using the `latin1` character set by default. If you have been storing column data that actually is in some other character set that the 4.1 server supports directly, you can convert the column. However, you should avoid trying to convert directly from `latin1` to the "real" character set. This may result in data loss. Instead, convert the column to a binary data type, and then from the binary type to a nonbinary type with the desired character set. Conversion to and from binary involves no attempt at

character value conversion and preserves your data intact. Suppose that you have a 4.0 table with three columns that are used to store values represented in `latin1`, `latin2`, and `utf8`:

```
CREATE TABLE t
(
    latin1_col CHAR(50),
    latin2_col CHAR(100),
    utf8_col CHAR(150)
);
```

For MySQL 4.1, you want to convert this table to leave `latin1_col` alone but change the `latin2_col` and `utf8_col` columns to have character sets of `latin2` and `utf8`. Before upgrading to 4.1, back up your table, then convert the columns as follows:

```
ALTER TABLE t MODIFY latin2_col BLOB;
ALTER TABLE t MODIFY utf8_col BLOB;
```

Then, after upgrading to 4.1, complete the conversion by issuing these statements:

```
ALTER TABLE t MODIFY latin2_col CHAR(100) CHARACTER SET latin2;
ALTER TABLE t MODIFY utf8_col CHAR(150) CHARACTER SET utf8;
```

The first two statements “remove” the character set information from the `latin2_col` and `utf8_col` columns. The second two statements assign the proper character sets to the two columns.

If you like, you can combine the to-binary conversions and from-binary conversions into single statements. In MySQL 4.0, do this:

```
ALTER TABLE t
    MODIFY latin2_col BLOB,
    MODIFY utf8_col BLOB;
```

After upgrading to 4.1, do this:

```
ALTER TABLE t
    MODIFY latin2_col CHAR(100) CHARACTER SET latin2,
    MODIFY utf8_col CHAR(150) CHARACTER SET utf8;
```

If you can ensure that the tables will not otherwise be modified before you perform the character set conversion, you can issue all of the `ALTER TABLE` statements after upgrading to MySQL 4.1.

If you specified attributes when creating a column initially, you should also specify them when altering the table with `ALTER TABLE`. For example, if you specified `NOT NULL` and an explicit `DEFAULT` value, you should also provide them in the `ALTER TABLE` statement. Otherwise, the resulting column definition will not include those attributes.

9.1.12 Character Sets and Collations That MySQL Supports

MySQL supports 70+ collations for 30+ character sets. This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the permissible collations are listed.

You can always list the available character sets and their default collations with the `SHOW CHARACTER SET` statement:

```
mysql> SHOW CHARACTER SET;
```

```
+----------+-----------------------------+---------------------+
| Charset  | Description                 | Default collation   |
+----------+-----------------------------+---------------------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |
| dec8     | DEC West European           | dec8_swedish_ci     |
| cp850    | DOS West European           | cp850_general_ci    |
| hp8      | HP West European            | hp8_english_ci      |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |
| latin1   | cp1252 West European        | latin1_swedish_ci   |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |
| ascii    | US ASCII                    | ascii_general_ci    |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |
| cp1250   | Windows Central European    | cp1250_general_ci   |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |
| cp866    | DOS Russian                 | cp866_general_ci    |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |
| macce    | Mac Central European        | macce_general_ci    |
| macroman | Mac West European           | macroman_general_ci |
| cp852    | DOS Central European        | cp852_general_ci    |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |
| cp1256   | Windows Arabic              | cp1256_general_ci   |
| cp1257   | Windows Baltic              | cp1257_general_ci   |
| binary   | Binary pseudo charset       | binary              |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |
| cp932    | SJIS for Windows Japanese   | cp932_japanese_ci   |
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |
+----------+-----------------------------+---------------------+
```

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

9.1.12.1 Unicode Character Sets

MySQL 4.1 has two Unicode character sets:

- `ucs2`, the UCS-2 encoding of the Unicode character set using 16 bits per character
- `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character

You can store text in about 650 languages using these character sets. This section lists the collations available for each Unicode character set and describes their differentiating properties. For general information about the character sets, see Section 9.1.9, “Unicode Support”.

A similar set of collations is available for each Unicode character set. These are shown in the following list, where *`xxx`* represents the character set name. For example, *`xxx`*`_danish_ci` represents the Danish collations, the specific names of which are `ucs2_danish_ci` and `utf8_danish_ci`.

- *xxx*_bin
- *xxx*_czech_ci
- *xxx*_danish_ci
- *xxx*_estonian_ci
- *xxx*_general_ci (default)
- *xxx*_icelandic_ci
- *xxx*_latvian_ci
- *xxx*_lithuanian_ci
- *xxx*_persian_ci
- *xxx*_polish_ci
- *xxx*_roman_ci
- *xxx*_romanian_ci
- *xxx*_slovak_ci
- *xxx*_slovenian_ci
- *xxx*_spanish_ci
- *xxx*_spanish2_ci
- *xxx*_swedish_ci
- *xxx*_turkish_ci
- *xxx*_unicode_ci

There is a limitation in MySQL 4.1 that results in two characters not being correctly handled when a user tries to change their case using LOWER() [798] or UPPER() [804]:

- LATIN SMALL LETTER DOTLESS i
- LATIN CAPITAL LETTER I WITH DOT ABOVE

Here are two workarounds for MySQL 4.1:

1. Use ucs2 if you have Turkish data.
2. Use these function calls:

```
CONVERT(LOWER(CONVERT(col USING ucs2)) USING utf8)
```

MySQL implements the *xxx*_unicode_ci collations according to the Unicode Collation Algorithm (UCA) described at http://www.unicode.org/reports/tr10/. The collation uses the version-4.0.0 UCA weight keys: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. Currently, the *xxx*_unicode_ci collations have only partial support for the Unicode Collation Algorithm. Some characters are not supported yet. Also,

combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo.

MySQL implements language-specific Unicode collations only if the ordering with *`xxx`*`_unicode_ci` does not work well for a language. Language-specific collations are UCA-based. They are derived from *`xxx`*`_unicode_ci` with additional language tailoring rules.

For any Unicode character set, operations performed using the *`xxx`*`_general_ci` collation are faster than those for the *`xxx`*`_unicode_ci` collation. For example, comparisons for the `utf8_general_ci` collation are faster, but slightly less correct, than comparisons for `utf8_unicode_ci`. The reason for this is that `utf8_unicode_ci` supports mappings such as expansions; that is, when one character compares as equal to combinations of other characters. For example, in German and some other languages "ß" is equal to "`ss`". `utf8_unicode_ci` also supports contractions and ignorable characters. `utf8_general_ci` is a legacy collation that does not support expansions, contractions, or ignorable characters. It can make only one-to-one comparisons between characters.

To further illustrate, the following equalities hold in both `utf8_general_ci` and `utf8_unicode_ci` (for the effect this has in comparisons or when doing searches, see Section 9.1.7.8, "Examples of the Effect of Collation"):

```
Ä = A
Ö = O
Ü = U
```

A difference between the collations is that this is true for `utf8_general_ci`:

```
ß = s
```

Whereas this is true for `utf8_unicode_ci`, which supports the German DIN-1 ordering (also known as dictionary order):

```
ß = ss
```

MySQL implements language-specific collations for the `utf8` character set only if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German dictionary order and French, so there is no need to create special `utf8` collations.

`utf8_general_ci` also is satisfactory for both German and French, except that "ß" is equal to "`s`", and not to "`ss`". If this is acceptable for your application, you should use `utf8_general_ci` because it is faster. Otherwise, use `utf8_unicode_ci` because it is more accurate.

`xxx``_swedish_ci` includes Swedish rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Y < Ö
```

The *`xxx`*`_spanish_ci` and *`xxx`*`_spanish2_ci` collations correspond to modern Spanish and traditional Spanish, respectively. In both collations, "ñ" (n-tilde) is a separate letter between "`n`" and "`o`". In addition, for traditional Spanish, "`ch`" is a separate letter between "`c`" and "`d`", and "`ll`" is a separate letter between "`l`" and "`m`"

In the *`xxx`*`_roman_ci` collations, `I` and `J` compare as equal, and `U` and `V` compare as equal.

For additional information about Unicode collations in MySQL, see Collation-Charts.Org (utf8).

9.1.12.2 West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- `ascii` (US ASCII) collations:
  - `ascii_bin`
  - `ascii_general_ci` (default)
- `cp850` (DOS West European) collations:
  - `cp850_bin`
  - `cp850_general_ci` (default)
- `dec8` (DEC Western European) collations:
  - `dec8_bin`
  - `dec8_swedish_ci` (default)
- `hp8` (HP Western European) collations:
  - `hp8_bin`
  - `hp8_english_ci` (default)
- `latin1` (cp1252 West European) collations:
  - `latin1_bin`
  - `latin1_danish_ci`
  - `latin1_general_ci`
  - `latin1_general_cs`
  - `latin1_german1_ci`
  - `latin1_german2_ci`
  - `latin1_spanish_ci`
  - `latin1_swedish_ci` (default)

  `latin1` is the default character set. MySQL's `latin1` is the same as the Windows `cp1252` character set. This means it is the same as the official `ISO 8859-1` or IANA (Internet Assigned Numbers Authority) `latin1`, except that IANA `latin1` treats the code points between `0x80` and `0x9f` as "undefined," whereas `cp1252`, and therefore MySQL's `latin1`, assign characters for those positions. For example, `0x80` is the Euro sign. For the "undefined" entries in `cp1252`, MySQL translates `0x81` to Unicode `0x0081`, `0x8d` to `0x008d`, `0x8f` to `0x008f`, `0x90` to `0x0090`, and `0x9d` to `0x009d`.

  The `latin1_swedish_ci` collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

The `latin1_german1_ci` and `latin1_german2_ci` collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the “dictionary collation” and DIN-2 is called the “phone book collation.” For an example of the effect this has in comparisons or when doing searches, see Section 9.1.7.8, “Examples of the Effect of Collation”.

- `latin1_german1_ci` (dictionary) rules:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- `latin1_german2_ci` (phone-book) rules:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

In the `latin1_spanish_ci` collation, “ñ” (n-tilde) is a separate letter between “n” and “o”.

- `macroman` (Mac West European) collations:
  - `macroman_bin`
  - `macroman_general_ci` (default)
- `swe7` (7bit Swedish) collations:
  - `swe7_bin`
  - `swe7_swedish_ci` (default)

For additional information about Western European collations in MySQL, see Collation-Charts.Org (ascii, cp850, dec8, hp8, latin1, macroman, swe7).

9.1.12.3 Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- `cp1250` (Windows Central European) collations:
  - `cp1250_bin`
  - `cp1250_croatian_ci`
  - `cp1250_czech_cs`
  - `cp1250_general_ci` (default)
- `cp852` (DOS Central European) collations:
  - `cp852_bin`
  - `cp852_general_ci` (default)

- `keybcs2` (DOS Kamenicky Czech-Slovak) collations:
  - `keybcs2_bin`
  - `keybcs2_general_ci` (default)
- `latin2` (ISO 8859-2 Central European) collations:
  - `latin2_bin`
  - `latin2_croatian_ci`
  - `latin2_czech_cs`
  - `latin2_general_ci` (default)
  - `latin2_hungarian_ci`
- `macce` (Mac Central European) collations:
  - `macce_bin`
  - `macce_general_ci` (default)

For additional information about Central European collations in MySQL, see Collation-Charts.Org (cp1250, cp852, keybcs2, latin2, macce).

9.1.12.4 South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- `armscii8` (ARMSCII-8 Armenian) collations:
  - `armscii8_bin`
  - `armscii8_general_ci` (default)
- `cp1256` (Windows Arabic) collations:
  - `cp1256_bin`
  - `cp1256_general_ci` (default)
- `geostd8` (GEOSTD8 Georgian) collations:
  - `geostd8_bin`
  - `geostd8_general_ci` (default)
- `greek` (ISO 8859-7 Greek) collations:
  - `greek_bin`
  - `greek_general_ci` (default)
- `hebrew` (ISO 8859-8 Hebrew) collations:
  - `hebrew_bin`

  - `hebrew_general_ci` (default)
- `latin5` (ISO 8859-9 Turkish) collations:
  - `latin5_bin`
  - `latin5_turkish_ci` (default)

For additional information about South European and Middle Eastern collations in MySQL, see Collation-Charts.Org (armscii8, cp1256, geostd8, greek, hebrew, latin5).

9.1.12.5 Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- `cp1257` (Windows Baltic) collations:
  - `cp1257_bin`
  - `cp1257_general_ci` (default)
  - `cp1257_lithuanian_ci`
- `latin7` (ISO 8859-13 Baltic) collations:
  - `latin7_bin`
  - `latin7_estonian_cs`
  - `latin7_general_ci` (default)
  - `latin7_general_cs`

For additional information about Baltic collations in MySQL, see Collation-Charts.Org (cp1257, latin7).

9.1.12.6 Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- `cp1251` (Windows Cyrillic) collations:
  - `cp1251_bin`
  - `cp1251_bulgarian_ci`
  - `cp1251_general_ci` (default)
  - `cp1251_general_cs`
  - `cp1251_ukrainian_ci`
- `cp866` (DOS Russian) collations:
  - `cp866_bin`
  - `cp866_general_ci` (default)

- `koi8r` (KOI8-R Relcom Russian) collations:
  - `koi8r_bin`
  - `koi8r_general_ci` (default)
- `koi8u` (KOI8-U Ukrainian) collations:
  - `koi8u_bin`
  - `koi8u_general_ci` (default)

For additional information about Cyrillic collations in MySQL, see Collation-Charts.Org (cp1251, cp866, koi8r, koi8u).).

9.1.12.7 Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See The `cp932` Character Set, for additional information about the `cp932` and `sjis` character sets.

- `big5` (Big5 Traditional Chinese) collations:
  - `big5_bin`
  - `big5_chinese_ci` (default)
- `cp932` (SJIS for Windows Japanese) collations:
  - `cp932_bin`
  - `cp932_japanese_ci` (default)
- `eucjpms` (UJIS for Windows Japanese) collations:
  - `eucjpms_bin`
  - `eucjpms_japanese_ci` (default)
- `euckr` (EUC-KR Korean) collations:
  - `euckr_bin`
  - `euckr_korean_ci` (default)
- `gb2312` (GB2312 Simplified Chinese) collations:
  - `gb2312_bin`
  - `gb2312_chinese_ci` (default)
- `gbk` (GBK Simplified Chinese) collations:
  - `gbk_bin`
  - `gbk_chinese_ci` (default)
- `sjis` (Shift-JIS Japanese) collations:

  - `sjis_bin`
  - `sjis_japanese_ci` (default)
- `tis620` (TIS620 Thai) collations:
  - `tis620_bin`
  - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
  - `ujis_bin`
  - `ujis_japanese_ci` (default)

The `big5_chinese_ci` collation sorts on number of strokes.

For additional information about Asian collations in MySQL, see Collation-Charts.Org (big5, cp932, eucjpms, euckr, gb2312, gbk, sjis, tis620, ujis).

The `cp932` Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See http://www.iana.org/assignments/character-sets.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected—IBM extended characters, and IBM extended characters.

Since MySQL 4.1, many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted using Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL `cp932` character set is designed to solve these problems. It is available as of MySQL 4.1.12.

Before MySQL 4.1, it was safe to use any version of “SHIFT JIS” in conjunction with the `sjis` character set. However, because MySQL supports character set conversion beginning with 4.1, it is important to separate IANA `Shift_JIS` and `cp932` into two different character sets because they provide different conversion rules.

How does cp932 differ from sjis?

The cp932 character set differs from sjis in the following ways:

- cp932 supports NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.
- Some cp932 characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to cp932, one of the code points must be selected. For this "round trip conversion," the rule recommended by Microsoft is used. (See http://support.microsoft.com/kb/170559/EN-US/.)

  The conversion rule works like this:

  - If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
  - If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
  - If the character is in both IBM selected characters and NEC selected—IBM extended characters, use the code point of IBM extended characters.

  The table shown at http://www.microsoft.com/globaldev/reference/dbcs/932.htm provides information about the Unicode values of cp932 characters. For cp932 table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (ucs2) encoding. For table entries with an underlined two-digit value appears, there is a range of cp932 character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the cp932 characters that begin with those digits.

  The following links are of special interest. They correspond to the encodings for the following sets of characters:

  - NEC special characters:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_87.htm
    ```

  - NEC selected—IBM extended characters:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_ED.htm
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_EE.htm
    ```

  - IBM selected characters:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_FA.htm
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_FB.htm
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_FC.htm
    ```

For some characters, conversion to and from ucs2 is different for sjis and cp932. The following tables illustrate these differences.

Conversion to ucs2:

sjis/cp932 Value	sjis -> ucs2 Conversion	cp932 -> ucs2 Conversion
5C	005C	005C

sjis/cp932 Value	sjis -> ucs2 Conversion	cp932 -> ucs2 Conversion
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from ucs2:

ucs2 value	ucs2 -> sjis Conversion	ucs2 -> cp932 Conversion
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` [385] (or `--skip-character-set-client-handshake` [385]) has an important effect. See Section 5.1.2, “Server Command Options”.

9.2 Using the German Character Set

In MySQL 4.0, to get German sorting order, you should start `mysqld` with a `--default-character-set=latin1_de` [385] option. This affects server behavior in several ways:

- When sorting and comparing strings, the following mapping is performed on the strings before doing the comparison:

```
ä  ->  ae
ö  ->  oe
ü  ->  ue
ß  ->  ss
```

- All accented characters are converted to their unaccented uppercase counterpart. All letters are converted to uppercase.
- When comparing strings with `LIKE` [804], the one-character to two-character mapping is not done. All letters are converted to uppercase. Accents are removed from all letters except Ü, ü, Ö, ö, Ä, and ä.

In MySQL 4.1 and up, character set and collation are specified separately. You should select the `latin1` character set and either the `latin1_german1_ci` or `latin1_german2_ci` collation. For example, to start the server with the `latin1_german1_ci` collation, use the `--character-set-server=latin1` [385] and `--collation-server=latin1_german1_ci` [385] options.

For information on the differences between these two collations, see Section 9.1.12.2, “West European Character Sets”.

9.3 Setting the Error Message Language

By default, `mysqld` produces error messages in English, but they can also be displayed in any of several other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish.

You can select which language the server uses for error messages using the instructions in this section.

To start `mysqld` with a particular language for error messages, use the `--language` [388] or `-L` option. The option value can be a language name or the full path to the error message file. For example:

```
shell> mysqld --language=swedish
```

Or:

```
shell> mysqld --language=/usr/local/share/swedish
```

The language name should be specified in lowercase.

By default, the language files are located in the `share/mysql/LANGUAGE` directory under the MySQL base directory.

For information about changing the character set for error messages (rather than the language), see Section 9.1.6, “Character Set for Error Messages”.

You can change the content of the error messages produced by the server using the instructions in the MySQL Internals manual, available at MySQL Internals: Error Messages. If you do change the content of error messages, remember to repeat your changes after each upgrade to a newer version of MySQL.

9.4 Adding a New Character Set

This section discusses the procedure for adding a new character set to MySQL. You must have a MySQL source distribution to use these instructions. There is one procedure for MySQL 4.1 and a different one for MySQL 4.0 or older. For either procedure, the instructions depend on whether the character set is simple or complex:

- If the character set does not need to use special string collating routines for sorting and does not need multi-byte character support, it is simple.
- If the character set needs either of those features, it is complex.

For example, `greek` and `swe7` are simple character sets, whereas `big5` and `czech` are complex character sets.

In the following instructions, *`MYSET`* represents the name of the character set that you want to add.

If you have MySQL 4.1, use this procedure to add a new character set:

1. Add a `<charset>` element for *`MYSET`* to the `sql/share/charsets/Index.xml` file. Use the existing contents in the file as a guide to adding new contents.

   The `<charset>` element must list all the collations for the character set. These must include at least a binary collation and a default collation. The default collation is usually named using a suffix of `general_ci` (general, case insensitive). It is possible for the binary collation to be the default collation, but usually they are different. The default collation should have a `primary` flag. The binary collation should have a `binary` flag.

   You must assign a unique ID number to each collation, chosen from the range 1 to 254. To see the currently used collation IDs, use this query:

   ```
   SHOW COLLATION;
   ```

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multi-byte functions, or both.

   For a simple character set, create a configuration file, *`MYSET`*`.xml`, that describes the character set properties. Create this file in the `sql/share/charsets` directory. (You can use a copy of `latin1.xml` as the basis for this file.) The syntax for the file is very simple:

   - Comments are written as ordinary XML comments (`<!-- text -->`).
   - Words within `<map>` array elements are separated by arbitrary amounts of whitespace.
   - Each word within `<map>` array elements must be a number in hexadecimal format.
   - The `<map>` array element for the `<ctype>` element has 257 words. The other `<map>` array elements after that have 256 words. See Section 9.4.1, “The Character Definition Arrays”.
   - For each collation listed in the `<charset>` element for the character set in `Index.xml`, *`MYSET`*`.xml` must contain a `<collation>` element that defines the character ordering.

   For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

   a. Create the file `ctype-`*`MYSET`*`.c` in the `strings` directory. Look at one of the existing `ctype-*.c` files (such as `ctype-big5.c`) to see what needs to be defined. The arrays in your file must have names like `ctype_`*`MYSET`*, `to_lower_`*`MYSET`*, and so on. These correspond to the arrays for a simple character set. See Section 9.4.1, “The Character Definition Arrays”.

   b. For each collation listed in the `<charset>` element for the character set in `Index.xml`, the `ctype-`*`MYSET`*`.c` file must provide an implementation of the collation.

c. If you need string collating functions, see Section 9.4.2, “String Collating Support”.

d. If you need multi-byte character support, see Section 9.4.3, “Multi-Byte Character Support”.

3. Follow these steps to modify the configuration information. Use the existing configuration information as a guide to adding information for *MYSYS*. The example here assumes that the character set has default and binary collations, but more lines will be needed if *MYSET* has additional collations.

   a. Edit `mysys/charset-def.c`, and “register” the collations for the new character set.

      Add these lines to the “declaration” section:

```
#ifdef HAVE_CHARSET_MYSET
extern CHARSET_INFO my_charset_MYSET_general_ci;
extern CHARSET_INFO my_charset_MYSET_bin;
#endif
```

      Add these lines to the “registration” section:

```
#ifdef HAVE_CHARSET_MYSET
  add_compiled_collation(&my_charset_MYSET_general_ci);
  add_compiled_collation(&my_charset_MYSET_bin);
#endif
```

   b. If the character set uses `ctype-MYSET.c`, edit `strings/Makefile.am` and add `ctype-MYSET.c` to each definition of the `CSRCS` variable, and to the `EXTRA_DIST` variable.

   c. If the character set uses `ctype-MYSET.c`, edit `libmysql/Makefile.shared` and add `ctype-MYSET.lo` to the `mystringsobjects` definition.

   d. Edit `configure.in`:

      i. Add *MYSET* to one of the `define(CHARSETS_AVAILABLE...)` lines in alphabetic order.

      ii. Add *MYSET* to `CHARSETS_COMPLEX`. This is needed even for simple character sets, or `configure` will not recognize `--with-charset=MYSET` [97].

      iii. Add *MYSET* to the first `case` control structure. Omit the `USE_MB` and `USE_MB_IDENT` lines for 8-bit character sets.

```
MYSET)
  AC_DEFINE(HAVE_CHARSET_MYSET, 1, [Define to enable charset MYSET])
  AC_DEFINE([USE_MB], 1, [Use multi-byte character routines])
  AC_DEFINE(USE_MB_IDENT, 1)
  ;;
```

      iv. Add *MYSET* to the second `case` control structure:

```
MYSET)
  default_charset_default_collation="MYSET_general_ci"
  default_charset_collations="MYSET_general_ci MYSET_bin"
  ;;
```

4. Reconfigure, recompile, and test.

If you have MySQL 4.0 or older, use this procedure to add a new character set:

1. Add *MYSET* to the end of the sql/share/charsets/Index file. Assign a unique number to it.

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multi-byte functions, or both.

   For a simple character set, create a configuration file that describes the character set properties. Create the file *MYSET*.conf file in the sql/share/charsets directory. (You can use a copy of latin1.conf as the basis for this file.) The syntax for the file is very simple:

   - Comments start with a "#" character and continue to the end of the line.
   - Words are separated by arbitrary amounts of whitespace.
   - When defining the character set, every word must be a number in hexadecimal format.
   - The ctype array takes up the first 257 words. The to_lower[], to_upper[], and sort_order[] arrays take up 256 words each after that. See Section 9.4.1, "The Character Definition Arrays".

   For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

   a. Create the file ctype-*MYSET*.c in the strings directory. Look at one of the existing ctype-*.c files (such as ctype-big5.c) to see what needs to be defined. The arrays in your file must have names like ctype_*MYSET*, to_lower_*MYSET*, and so on. These correspond to the arrays for a simple character set. See Section 9.4.1, "The Character Definition Arrays".

   b. Near the top of the file, place a special comment like this:

```
/*
 * This comment is parsed by configure to create ctype.c,
 * so don't change it unless you know what you are doing.
 *
 * .configure. strxfrm_multiply_MYSET=N
 * .configure. mbmaxlen_MYSET=N
 */
```

      The configure program uses this comment to include the character set into the MySQL library automatically.

      If you need string collating functions, you must specify the strxfrm_multiply_*MYSET*=*N* value in the special comment at the top of the source file. *N* must be a positive integer that indicates the maximum ratio to which strings may grow during execution of the my_strxfrm_*MYSET*() function.

      If you need multi-byte character set functions, you must specify the mbmaxlen_*MYSET*=*N* value in the special comment at the top of the ctype-*MYSET*.c source file for your character set. *N* should be set to the size in bytes of the largest character in the set.

   c. If you need string collating functions, see Section 9.4.2, "String Collating Support".

   d. If you need multi-byte character support, see Section 9.4.3, "Multi-Byte Character Support".

3. Follow these steps to modify the configuration information. Use the existing configuration information as a guide to adding information for *MYSYS*.

   a. Add the character set name to the CHARSETS_AVAILABLE list in configure.in.

   b. If the character set uses `ctype-MYSET.c`, edit `strings/Makefile.am` and add `ctype-MYSET.c` to the `EXTRA_DIST` variable.

4. Reconfigure, recompile, and test.

The `sql/share/charsets/README` file includes additional instructions.

9.4.1 The Character Definition Arrays

Each simple character set has a configuration file located in the `sql/share/charsets` directory. The file is named `MYSET.xml` for MySQL 4.1 and `MYSET.conf` for MySQL 4.0 or older.

For MySQL 4.1, `MYSET.xml` files use `<map>` array elements to list character set properties. `<map>` elements appear within these elements:

- `<ctype>` defines attributes for each character
- `<lower>` and `<upper>` list the lowercase and uppercase characters
- `<unicode>` maps 8-bit character values to Unicode values
- `<collation>` elements indicate character ordering for comparisons and sorts, one element per collation (binary collations need no `<map>` element because the character codes themselves provide the ordering)

For MySQL 4.0 or older, `MYSET.conf` files list character set properties using these arrays:

- `ctype[]` defines attributes for each character
- `to_lower[]` and `to_upper[]` list the lowercase and uppercase characters
- `sort_order[]` indicates character ordering for comparisons and sorts

For a complex character set as implemented in a `ctype-MYSET.c` file in the `strings` directory, there are corresponding arrays: `ctype_MYSET[]`, `to_lower_MYSET[]`, and so forth. Not every complex character set has all of the arrays. See the existing `ctype-*.c` files for examples. For MySQL 4.1, see the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

The `ctype` array is indexed by character value + 1 and has 257 elements. This is an old legacy convention for handling `EOF`. The other arrays are indexed by character value and have 256 elements.

`ctype` array elements are bit values. Each element describes the attributes of a single character in the character set. Each attribute is associated with a bitmask, as defined in `include/m_ctype.h`:

```
#define _MY_U   01      /* Upper case */
#define _MY_L   02      /* Lower case */
#define _MY_NMR 04      /* Numeral (digit) */
#define _MY_SPC 010     /* Spacing character */
#define _MY_PNT 020     /* Punctuation */
#define _MY_CTR 040     /* Control character */
#define _MY_B   0100    /* Blank */
#define _MY_X   0200    /* heXadecimal digit */
```

The `ctype` value for a given character should be the union of the applicable bitmask values that describe the character. For example, `'A'` is an uppercase character (`_MY_U`) as well as a hexadecimal digit (`_MY_X`), so its `ctype` value should be defined like this:

```
ctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

The bitmask values in `m_ctype.h` are octal values, but the elements of the `ctype` array in *`MYSET`*`.xml` (or *`MYSET`*`.conf`) should be written as hexadecimal values.

The `lower` and `upper` (or `to_lower` and `to_upper`) arrays hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

Each `collation` (or `sort_order`) array is a map indicating how characters should be ordered for comparison and sorting purposes. MySQL sorts characters based on the values of this information. In some cases, this is the same as the `upper` array, which means that sorting is case-insensitive. For more complicated sorting rules (for complex character sets), see the discussion of string collating in Section 9.4.2, “String Collating Support”.

9.4.2 String Collating Support

For simple character sets in MySQL 4.1, sorting rules are specified in the *`MYSET`*`.xml` configuration file using `<map>` array elements within `<collation>` elements. (For MySQL 4.0 or older, the rules are given by the `sort_order` array in *`MYSET`*`.conf`.) If the sorting rules for your language are too complex to be handled with simple arrays, you need to define string collating functions in the `ctype-`*`MYSET`*`.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets. For MySQL 4.1, take a look at the `MY_COLLATION_HANDLER` structures to see how they are used, and see the `CHARSET_INFO.txt` file in the `strings` directory for additional information. For MySQL 4.0 or older, look at the `CHARSET_INFO` structures.

9.4.3 Multi-Byte Character Support

If you want to add support for a new character set that includes multi-byte characters, you need to use multi-byte character functions in the `ctype-`*`MYSET`*`.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets. For MySQL 4.1, take a look at the `MY_CHARSET_HANDLER` structures to see how they are used, and see the `CHARSET_INFO.txt` file in the `strings` directory for additional information. For MySQL 4.0 or older, look at the `CHARSET_INFO` structures.

9.5 How to Add a New Collation to a Character Set

A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

A collation orders characters based on weights. Each character in a character set maps to a weight. Characters with equal weights compare as equal, and characters with unequal weights compare according to the relative magnitude of their weights.

MySQL supports several collation implementations, as discussed in Section 9.5.1, “Collation Implementation Types”. Some of these can be added to MySQL without recompiling:

- Simple collations for 8-bit character sets

- UCA-based collations for Unicode character sets
- Binary (*xxx*_bin) collations

The following discussion describes how to add simple collations for 8-bit character sets. In MySQL 4.1, this is the only type of collation that can be added without recompiling. To add a UCA-based Unicode collation, MySQL 5.0 or higher is required.

All existing character sets already have a binary collation, so there is no need here to describe how to add one.

Summary of the procedure for adding a new collation:

1. Choose a collation ID
2. Add configuration information that names the collation and describes the character-ordering rules
3. Restart the server
4. Verify that the collation is present

The instructions here cover only collations that can be added without recompiling MySQL. To add a collation that does require recompiling (as implemented by means of functions in a C source file), use the instructions in Section 9.4, “Adding a New Character Set”. However, instead of adding all the information required for a complete character set, just modify the appropriate files for an existing character set. That is, based on what is already present for the character set's current collations, add new data structures, functions, and configuration information for the new collation. For an example, see the MySQL Blog article in the following list of additional resources.

Additional Resources

- The Unicode Collation Algorithm (UCA) specification: http://www.unicode.org/reports/tr10/
- The Locale Data Markup Language (LDML) specification: http://www.unicode.org/reports/tr35/
- MySQL Blog article “Instructions for adding a new Unicode collation”: http://blogs.mysql.com/peterg/2008/05/19/instructions-for-adding-a-new-unicode-collation/

9.5.1 Collation Implementation Types

MySQL implements several types of collations:

Simple collations for 8-bit character sets

This kind of collation is implemented using an array of 256 weights that defines a one-to-one mapping from character codes to weights. latin1_swedish_ci is an example. It is a case-insensitive collation, so the uppercase and lowercase versions of a character have the same weights and they compare as equal.

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)
```

Complex collations for 8-bit character sets

This kind of collation is implemented using functions in a C source file that define how to order characters, as described in Section 9.4, “Adding a New Character Set”.

Collations for non-Unicode multi-byte character sets

For this type of collation, 8-bit (single-byte) and multi-byte characters are handled differently. For 8-bit characters, character codes map to weights in case-insensitive fashion. (For example, the single-byte characters `'a'` and `'A'` both have a weight of `0x41`.) For multi-byte characters, there are two types of relationship between character codes and weights:

- Weights equal character codes. `sjis_japanese_ci` is an example of this kind of collation. The multi-byte character `'ぢ'` has a character code of `0x82C0`, and the weight is also `0x82C0`.
- Character codes map one-to-one to weights, but a code is not necessarily equal to the weight. `gbk_chinese_ci` is an example of this kind of collation. The multi-byte character `'膰'` has a character code of `0x81B0` but a weight of `0xC286`.

Collations for Unicode multi-byte character sets

Some of these collations are based on the Unicode Collation Algorithm (UCA), others are not.

Non-UCA collations have a one-to-one mapping from character code to weight. In MySQL, such collations are case insensitive and accent insensitive. `utf8_general_ci` is an example: `'a'`, `'A'`, `'À'`, and `'á'` each have different character codes but all have a weight of `0x0041` and compare as equal.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a' = 'A', 'a' = 'À', 'a' = 'á';
+-----------+-----------+-----------+
| 'a' = 'A' | 'a' = 'À' | 'a' = 'á' |
+-----------+-----------+-----------+
|         1 |         1 |         1 |
+-----------+-----------+-----------+
1 row in set (0.06 sec)
```

UCA-based collations in MySQL have these properties:

- If a character has weights, each weight uses 2 bytes (16 bits)
- A character may have zero weights (or an empty weight). In this case, the character is ignorable. Example: "U+0000 NULL" does not have a weight and is ignorable.
- A character may have one weight. Example: `'a'` has a weight of `0x0E33`.
- A character may have many weights. This is an expansion. Example: The German letter `'ß'` (SZ ligature, or SHARP S) has a weight of `0x0FEA0FEA`.
- Many characters may have one weight. This is a contraction. Example: `'ch'` is a single letter in Czech and has a weight of `0x0EE2`.

A many-characters-to-many-weights mapping is also possible (this is contraction with expansion), but is not supported by MySQL.

Miscellaneous collations

There are also a few collations that do not fall into any of the previous categories.

9.5.2 Choosing a Collation ID

Each collation must have a unique ID. To add a new collation, you must choose an ID value that is not currently used. The value must be in the range from 1 to 254. The collation ID that you choose will show up in these contexts:

- The `Id` column of `SHOW COLLATION` output
- The `charsetnr` member of the `MYSQL_FIELD` C API data structure

To display a list of the currently used collation IDs, use this statement:

```
mysql> SHOW COLLATION;
+----------------------+----------+-----+---------+----------+---------+
| Collation            | Charset  | Id  | Default | Compiled | Sortlen |
+----------------------+----------+-----+---------+----------+---------+
| big5_chinese_ci      | big5     |   1 | Yes     | Yes      |       1 |
| big5_bin             | big5     |  84 |         | Yes      |       1 |
...
| latin1_german1_ci    | latin1   |   5 |         | Yes      |       1 |
| latin1_swedish_ci    | latin1   |   8 | Yes     | Yes      |       1 |
| latin1_danish_ci     | latin1   |  15 |         | Yes      |       1 |
| latin1_german2_ci    | latin1   |  31 |         | Yes      |       2 |
| latin1_bin           | latin1   |  47 |         | Yes      |       1 |
| latin1_general_ci    | latin1   |  48 |         | Yes      |       1 |
| latin1_general_cs    | latin1   |  49 |         | Yes      |       1 |
| latin1_spanish_ci    | latin1   |  94 |         | Yes      |       1 |
| latin2_czech_cs      | latin2   |   2 |         | Yes      |       4 |
| latin2_general_ci    | latin2   |   9 | Yes     | Yes      |       1 |
| latin2_hungarian_ci  | latin2   |  21 |         | Yes      |       1 |
| latin2_croatian_ci   | latin2   |  27 |         | Yes      |       1 |
| latin2_bin           | latin2   |  77 |         | Yes      |       1 |
...
+----------------------+----------+-----+---------+----------+---------+
```

Look through the values in the `Id` column and pick a value that is not used.

Warning

If you upgrade MySQL, you may find that the collation ID you choose has been assigned to a collation included in the new MySQL distribution. In this case, you will need to choose a new value for your own collation.

In addition, before upgrading, you should save the configuration files that you change. If you upgrade in place, the process will replace the your modified files.

9.5.3 Adding a Simple Collation to an 8-Bit Character Set

To add a simple collation for an 8-bit character set without recompiling MySQL, use the following procedure. The example adds a collation named `latin1_test_ci` to the `latin1` character set.

1. Choose a collation ID, as shown in Section 9.5.2, “Choosing a Collation ID”. The following steps use an ID of 56.
2. You will need to modify the `Index.xml` and `latin1.xml` configuration files. These files will be located in the directory named by the `character_sets_dir` [409] system variable. You can check the variable value as follows, although the path name might be different on your system:

   ```
   mysql> SHOW VARIABLES LIKE 'character_sets_dir';
   ```

```
+--------------------+----------------------------------------+
| Variable_name      | Value                                  |
+--------------------+----------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+----------------------------------------+
```

3. Choose a name for the collation and list it in the `Index.xml` file. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID. For example:

```
<charset name="latin1">
  ...
  <!-- associate collation name with its ID -->
  <collation name="latin1_test_ci" id="56"/>
  ...
</charset>
```

4. In the `latin1.xml` configuration file, add a `<collation>` element that names the collation and that contains a `<map>` element that defines a character code-to-weight mapping table. Each word within the `<map>` element must be a number in hexadecimal format.

```
<collation name="latin1_test_ci">
<map>
 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
 60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C D7 5C 55 55 55 59 59 DE DF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C F7 5C 55 55 55 59 59 DE FF
</map>
</collation>
```

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION LIKE 'latin1_test_ci';
+----------------+---------+----+---------+----------+---------+
| Collation      | Charset | Id | Default | Compiled | Sortlen |
+----------------+---------+----+---------+----------+---------+
| latin1_test_ci | latin1  | 56 |         |          |       1 |
+----------------+---------+----+---------+----------+---------+
```

9.6 Character Set Configuration

You can change the default server character set and collation with the `--character-set-server` [385] and `--collation-server` [385] options when you start the server. The collation must be a legal collation for the default character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.) See Section 5.1.2, “Server Command Options”.

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- Your program uses an incorrect path to determine where the character sets are stored (which is typically the `share/mysql/charsets` or `share/charsets` directory under the MySQL installation directory). This can be fixed by using the `--character-sets-dir` option when you run the program in question. For example, to specify a directory to be used by MySQL client programs, list it in the `[client]` group of your option file. The examples given here show what the setting might look like for Unix or Windows, respectively:

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets

[client]
character-sets-dir="C:/Program Files/MySQL/MySQL Server 4.1/share/charsets"
```

- The character set is a complex character set that cannot be loaded dynamically. In this case, you must recompile the program with support for the character set.
- The character set is a dynamic character set, but you do not have a configuration file for it. In this case, you should install the configuration file for the character set from a new MySQL distribution.
- If your character set index file does not contain the name for the character set, your program displays an error message. In MySQL 4.1, the file is named `Index.xml` and the message is:

```
Character set 'charset_name' is not a compiled character set and is not
specified in the '/usr/share/mysql/charsets/Index.xml' file
```

  Before MySQL 4.1, the file is named `Index` and the message is:

```
ERROR 1105: File '/usr/local/share/mysql/charsets/charset_name.conf'
not found (Errcode: 2)
```

  To solve this problem, you should either get a new index file or manually add the name of any missing character sets to the current file.

You can force client programs to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary. However, when `character_set_system` [409] differs from `character_set_server` [408] or `character_set_client` [408], and you input characters manually (as database object identifiers, column values, or both), these may be displayed incorrectly in output from the client or the output itself may be formatted incorrectly. In such cases, starting the mysql client with `--default-character-set=system_character_set` [260]—that is, setting the client character set to match the system character set—should fix the problem.

For `MyISAM` tables, you can check the character set name and number for a table with `myisamchk -dvv tbl_name`.

9.7 MySQL Server Time Zone Support

You can set the system time zone for MySQL Server at startup with the `--timezone=timezone_name` [245] option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`. The permissible values for `--timezone` [245] or `TZ` are system-dependent. Consult your operating system documentation to see what values are acceptable.

Before MySQL 4.1.3, the server operates only in the system time zone set at startup. Beginning with MySQL 4.1.3, the server maintains several time zone settings, some of which can be modified at runtime:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` [430] system variable. The value does not change thereafter.
- The server's current time zone. The global `time_zone` [431] system variable indicates the time zone the server currently is operating in. The initial value for `time_zone` [431] is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone.

  The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone=timezone` [386] option on the command line, or you can use the following line in an option file:

  ```
  default-time-zone='timezone'
  ```

  If you have the `SUPER` [493] privilege, you can set the global server time zone value at runtime with this statement:

  ```
  mysql> SET GLOBAL time_zone = timezone;
  ```

- Per-connection time zones. Each client that connects has its own time zone setting, given by the session `time_zone` [431] variable. Initially, the session variable takes its value from the global `time_zone` [431] variable, but the client can change its own time zone with this statement:

  ```
  mysql> SET time_zone = timezone;
  ```

The current session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` [837] or `CURTIME()` [829], and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the current time zone to UTC for storage, and from UTC to the current time zone for retrieval.

The current time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` [843] or values in `DATE`, `TIME`, or `DATETIME` columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from `TIMESTAMP` values. If you want locale-specific arithmetic for `DATE`, `TIME`, or `DATETIME` values, convert them to UTC, perform the arithmetic, and then convert back.

The current values of the global and client-specific time zones can be retrieved like this:

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

`timezone` values can be given in several formats, none of which are case sensitive:

- The value `'SYSTEM'` indicates that the time zone should be the same as the system time zone.
- The value can be given as a string indicating an offset from UTC, such as `'+10:00'` or `'-6:00'`.
- The value can be given as a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, or `'MET'`. Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated.

The MySQL installation procedure creates the time zone tables in the `mysql` database, but does not load them. You must do so manually using the following instructions. (If you are upgrading to MySQL 4.1.3 or later from an earlier version, you can create the tables by upgrading your `mysql` database. Use the

instructions in Section 4.4.5, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”. After creating the tables, you can load them.)

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. For example, the rules for Daylight Saving Time in the United States, Mexico, and parts of Canada changed in 2007. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See the notes at the end of this section.

If your system has its own *zoneinfo database* (the set of files describing time zones), you should use the `mysql_tzinfo_to_sql` program for filling the time zone tables. Examples of such systems are Linux, FreeBSD, Sun Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a zoneinfo database, you can use the downloadable package described later in this section.

The `mysql_tzinfo_to_sql` program is used to load the time zone tables. On the command line, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or to generate leap second information:

- To load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*, invoke `mysql_tzinfo_to_sql` like this:

  ```
  shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
  ```

  With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone needs to account for leap seconds, initialize the leap second information like this, where *`tz_file`* is the name of your time zone file:

  ```
  shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
  ```

- After running `mysql_tzinfo_to_sql`, it is best to restart the server so that it does not continue to use any previously cached time zone data.

If your system is one that has no zoneinfo database (for example, Windows or HP-UX), you can use the package of pre-built time zone tables that is available for download at the MySQL Developer Zone:

```
http://dev.mysql.com/downloads/timezones.html
```

This time zone package contains `.frm`, `.MYD`, and `.MYI` files for the `MyISAM` time zone tables. These tables should be part of the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. The server should be stopped while you do this and restarted afterward.

Warning

Do not use the downloadable package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

For information about time zone settings in replication setup, please see Section 14.7, “Replication Features and Issues”.

9.7.1 Staying Current with Time Zone Changes

As mentioned earlier, when the time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are two factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the Web site for your operating system vendor for an update that addresses the time changes.
- If you replace the system's `/etc/localtime` timezone file with a version that uses rules differing from those in effect at `mysqld` startup, you should restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.
- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date. If your system has its own zoneinfo database, you should reload the MySQL time zone tables whenever the zoneinfo database is updated, using the instructions given earlier in this section. For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace your current time zone tables. `mysqld` caches time zone information that it looks up, so after replacing the time zone tables, you should restart `mysqld` to make sure that it does not continue to serve outdated time zone data.
- For versions of MySQL older than 4.1.3 that do not have time zone support, the server always tracks the operating system time (much like a time zone setting of `SYSTEM` in 4.1.3 and up). Assuming that the server host itself has its operating system updated to handle any changes to Daylight Saving Time rules, the MySQL server should know the correct time.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
```

A count of zero indicates that the table is empty. In this case, no one can be using named time zones, and you don't need to update the tables. A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, you should be sure to reload your time zone tables so that anyone who uses named time zones will get correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses these two queries:

```
SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you would see an incorrect result like this:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 01:00:00                                       |
+-----------------------------------------------------------+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 02:00:00                                       |
+-----------------------------------------------------------+
```

After updating the tables, you should see the correct result:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 01:00:00                                       |
+-----------------------------------------------------------+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+-----------------------------------------------------------+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
| 2007-03-11 01:00:00                                       |
+-----------------------------------------------------------+
```

9.8 MySQL Server Locale Support

Beginning with MySQL 4.1.21, the locale indicated by the `lc_time_names` [417] system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()` [832], `DAYNAME()` [833], and `MONTHNAME()` [837] functions.

Locale names have language and region subtags listed by IANA (http://www.iana.org/assignments/language-subtag-registry) such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting, but any client can examine or set its `lc_time_names` [417] value as shown in the following example:

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+----------------------+------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+----------------------+------------------------+
| Friday                | January                |
+----------------------+------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----------------------------------------+
| Friday Fri January Jan                  |
+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| es_MX           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+----------------------+------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+----------------------+------------------------+
| viernes               | enero                  |
+----------------------+------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----------------------------------------+
| viernes vie enero ene                   |
+-----------------------------------------+
1 row in set (0.00 sec)
```

The day or month name for each of the affected functions is converted from `utf8` to the character set indicated by the `character_set_connection` [408] system variable.

`lc_time_names` [417] may be set to any of the following locale values. The set of locales supported by MySQL may differ from those supported by your operating system.

`ar_AE`: Arabic - United Arab Emirates	`ar_BH`: Arabic - Bahrain
`ar_DZ`: Arabic - Algeria	`ar_EG`: Arabic - Egypt
`ar_IN`: Arabic - India	`ar_IQ`: Arabic - Iraq
`ar_JO`: Arabic - Jordan	`ar_KW`: Arabic - Kuwait
`ar_LB`: Arabic - Lebanon	`ar_LY`: Arabic - Libya
`ar_MA`: Arabic - Morocco	`ar_OM`: Arabic - Oman
`ar_QA`: Arabic - Qatar	`ar_SA`: Arabic - Saudi Arabia
`ar_SD`: Arabic - Sudan	`ar_SY`: Arabic - Syria
`ar_TN`: Arabic - Tunisia	`ar_YE`: Arabic - Yemen
`be_BY`: Belarusian - Belarus	`bg_BG`: Bulgarian - Bulgaria

ca_ES: Catalan - Spain	cs_CZ: Czech - Czech Republic
da_DK: Danish - Denmark	de_AT: German - Austria
de_BE: German - Belgium	de_CH: German - Switzerland
de_DE: German - Germany	de_LU: German - Luxembourg
EE: Estonian - Estonia	en_AU: English - Australia
en_CA: English - Canada	en_GB: English - United Kingdom
en_IN: English - India	en_NZ: English - New Zealand
en_PH: English - Philippines	en_US: English - United States
en_ZA: English - South Africa	en_ZW: English - Zimbabwe
es_AR: Spanish - Argentina	es_BO: Spanish - Bolivia
es_CL: Spanish - Chile	es_CO: Spanish - Columbia
es_CR: Spanish - Costa Rica	es_DO: Spanish - Dominican Republic
es_EC: Spanish - Ecuador	es_ES: Spanish - Spain
es_GT: Spanish - Guatemala	es_HN: Spanish - Honduras
es_MX: Spanish - Mexico	es_NI: Spanish - Nicaragua
es_PA: Spanish - Panama	es_PE: Spanish - Peru
es_PR: Spanish - Puerto Rico	es_PY: Spanish - Paraguay
es_SV: Spanish - El Salvador	es_US: Spanish - United States
es_UY: Spanish - Uruguay	es_VE: Spanish - Venezuela
eu_ES: Basque - Basque	fi_FI: Finnish - Finland
fo_FO: Faroese - Faroe Islands	fr_BE: French - Belgium
fr_CA: French - Canada	fr_CH: French - Switzerland
fr_FR: French - France	fr_LU: French - Luxembourg
gl_ES: Galician - Spain	gu_IN: Gujarati - India
he_IL: Hebrew - Israel	hi_IN: Hindi - India
hr_HR: Croatian - Croatia	hu_HU: Hungarian - Hungary
id_ID: Indonesian - Indonesia	is_IS: Icelandic - Iceland
it_CH: Italian - Switzerland	it_IT: Italian - Italy
ja_JP: Japanese - Japan	ko_KR: Korean - Republic of Korea
lt_LT: Lithuanian - Lithuania	lv_LV: Latvian - Latvia
mk_MK: Macedonian - FYROM	mn_MN: Mongolia - Mongolian
ms_MY: Malay - Malaysia	nb_NO: Norwegian(Bokmål) - Norway
nl_BE: Dutch - Belgium	nl_NL: Dutch - The Netherlands
no_NO: Norwegian - Norway	pl_PL: Polish - Poland
pt_BR: Portugese - Brazil	pt_PT: Portugese - Portugal
ro_RO: Romanian - Romania	ru_RU: Russian - Russia
ru_UA: Russian - Ukraine	sk_SK: Slovak - Slovakia
sl_SI: Slovenian - Slovenia	sq_AL: Albanian - Albania
sr_YU: Serbian - Yugoslavia	sv_FI: Swedish - Finland

sv_SE: Swedish - Sweden	ta_IN: Tamil - India
te_IN: Telugu - India	th_TH: Thai - Thailand
tr_TR: Turkish - Turkey	uk_UA: Ukrainian - Ukraine
ur_PK: Urdu - Pakistan	vi_VN: Vietnamese - Viet Nam
zh_CN: Chinese - China	zh_HK: Chinese - Hong Kong
zh_TW: Chinese - Taiwan Province of China	

lc_time_names [417] currently does not affect the STR_TO_DATE() [838] or GET_FORMAT() [835] function.

Chapter 10 Data Types

Table of Contents

MySQL supports a number of data types in several categories: numeric types, date and time types, and string (character and byte) types. This chapter provides an overview of these data types, a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the permissible formats in which you can specify values.

MySQL 4.1 and up also supports extensions for handing spatial data. For information about these data types, see Chapter 16, *Spatial Extensions*.

Data type descriptions use these conventions:

- *M* indicates the maximum display width for integer types. For floating-point and fixed-point types, *M* is the total number of digits that can be stored (the precision). For string types, *M* is the maximum length. The maximum permissible value of *M* depends on the data type.
- *D* applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale). The maximum possible value is 30, but should be no greater than *M*–2.
- Square brackets (“`[`” and “`]`”) indicate optional parts of type definitions.

10.1 Data Type Overview

10.1.1 Numeric Type Overview

A summary of the numeric data types follows. For additional information about properties and storage requirements of the numeric types, see Section 10.2, “Numeric Types”, and Section 10.5, “Data Type Storage Requirements”.

M indicates the maximum display width for integer types. The maximum legal display width is 255. Display width is unrelated to the range of values a type can contain, as described in Section 10.2, “Numeric Types”. For floating-point and fixed-point types, *M* is the total number of digits that can be stored.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Numeric data types that permit the `UNSIGNED` attribute also permit `SIGNED`. However, these data types are signed by default, so the `SIGNED` attribute has no effect.

As of MySQL 4.1, `SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

`SERIAL DEFAULT VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

Warning

When you use subtraction between integer values where one is of type `UNSIGNED`, the result is unsigned unless the `NO_UNSIGNED_SUBTRACTION` [459] SQL mode is enabled. See Section 11.10, “Cast Functions and Operators”.

- `BIT`

  In versions of MySQL up to and lincluding 4.1, `BIT` is a synonym for `TINYINT(1)`.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

  A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

- `BOOL`, `BOOLEAN`

  These types are synonyms for `TINYINT(1)`. The synonym `BOOLEAN` was added in MySQL 4.1.0. A value of zero is considered false. Nonzero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+-----------------------+
| IF(0, 'true', 'false') |
+-----------------------+
| false                 |
+-----------------------+

mysql> SELECT IF(1, 'true', 'false');
+-----------------------+
| IF(1, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+

mysql> SELECT IF(2, 'true', 'false');
+-----------------------+
| IF(2, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+
```

However, the values TRUE and FALSE are merely aliases for 1 and 0, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+--------------------------------+
| IF(0 = FALSE, 'true', 'false') |
+--------------------------------+
| true                           |
+--------------------------------+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-------------------------------+
| IF(1 = TRUE, 'true', 'false') |
+-------------------------------+
| true                          |
+-------------------------------+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-------------------------------+
| IF(2 = TRUE, 'true', 'false') |
+-------------------------------+
| false                         |
+-------------------------------+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+--------------------------------+
| IF(2 = FALSE, 'true', 'false') |
+--------------------------------+
| false                          |
+--------------------------------+
```

The last two statements display the results shown because 2 is equal to neither 1 nor 0.

- SMALLINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A small integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.

- MEDIUMINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.

- INT[(*M*)] [UNSIGNED] [ZEROFILL]

  A normal-size integer. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295.

- INTEGER[(*M*)] [UNSIGNED] [ZEROFILL]

  This type is a synonym for INT.

- BIGINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A large integer. The signed range is -9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615.

  As of MySQL 4.1, SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

  Some things you should be aware of with respect to BIGINT columns:

  - All arithmetic is done using signed BIGINT or DOUBLE values, so you should not use unsigned big integers larger than 9223372036854775807 (63 bits) except with bit functions! If you do that, some

of the last digits in the result may be wrong because of rounding errors when converting a `BIGINT` value to a `DOUBLE`.

MySQL 4.0 can handle `BIGINT` in the following cases:

- When using integers to store large unsigned values in a `BIGINT` column.
- In `MIN(col_name)` [884] or `MAX(col_name)` [884], where `col_name` refers to a `BIGINT` column.
- When using operators (`+` [815], `-` [815], `*` [816], and so on) where both operands are integers.

- You can always store an exact integer value in a `BIGINT` column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
- The `-` [815], `+` [815], and `*` [816] operators use `BIGINT` arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than `9223372036854775807`.

- `DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]`

  An unpacked fixed-point number. Behaves like a `CHAR` column; “unpacked” means the number is stored as a string, using one character for each digit of the value. `M` is the total number of digits and `D` is the number of digits after the decimal point. The decimal point and (for negative numbers) the “`-`” sign are not counted in `M`, although space for them is reserved. If `D` is 0, values have no decimal point or fractional part. The maximum range of `DECIMAL` values is the same as for `DOUBLE`, but the actual range for a given `DECIMAL` column may be constrained by the choice of `M` and `D`. If `D` is omitted, the default is 0. If `M` is omitted, the default is 10.

  `UNSIGNED`, if specified, disallows negative values.

  Note

  Before MySQL 3.23, the value of `M` must be large enough to include the space needed for the sign and the decimal point characters.

- `DEC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]`

  These types are synonyms for `DECIMAL`. The `FIXED` synonym was added in MySQL 4.1.0 for compatibility with other database systems.

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

  A small (single-precision) floating-point number. Permissible values are `-3.402823466E+38` to `-1.175494351E-38`, `0`, and `1.175494351E-38` to `3.402823466E+38`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  `M` is the total number of digits and `D` is the number of digits following the decimal point. If `M` and `D` are omitted, values are stored to the limits permitted by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

  `UNSIGNED`, if specified, disallows negative values.

Using FLOAT might give you some unexpected problems because all calculations in MySQL are done with double precision. See Section B.5.5.7, “Solving Problems with No Matching Rows”.

- DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

  A normal-size (double-precision) floating-point number. Permissible values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

  UNSIGNED, if specified, disallows negative values.

- DOUBLE PRECISION[(*M*,*D*)] [UNSIGNED] [ZEROFILL], REAL[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

  These types are synonyms for DOUBLE. Exception: If the REAL_AS_FLOAT [460] SQL mode is enabled, REAL is a synonym for FLOAT rather than DOUBLE.

- FLOAT(*p*) [UNSIGNED] [ZEROFILL]

  A floating-point number. *p* represents the precision in bits, but MySQL uses this value only to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT with no *M* or *D* values. If *p* is from 25 to 53, the data type becomes DOUBLE with no *M* or *D* values. The range of the resulting column is the same as for the single-precision FLOAT or double-precision DOUBLE data types described earlier in this section.

  As of MySQL 3.23, this data type holds true floating-point values. In earlier MySQL versions, FLOAT(*p*) always has two decimals.

  FLOAT(*p*) syntax is provided for ODBC compatibility.

10.1.2 Date and Time Type Overview

A summary of the temporal data types follows. For additional information about properties and storage requirements of the temporal types, see Section 10.3, “Date and Time Types”, and Section 10.5, “Data Type Storage Requirements”. For descriptions of functions that operate on temporal values, see Section 11.7, “Date and Time Functions”.

For the DATE and DATETIME range descriptions, “supported” means that although earlier values might work, there is no guarantee.

- DATE

  A date. The supported range is '1000-01-01' to '9999-12-31'. MySQL displays DATE values in 'YYYY-MM-DD' format, but permits assignment of values to DATE columns using either strings or numbers.

- DATETIME

A date and time combination. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`. MySQL displays `DATETIME` values in `'YYYY-MM-DD HH:MM:SS'` format, but permits assignment of values to `DATETIME` columns using either strings or numbers.

- `TIMESTAMP[(M)]`

  A timestamp. The range is `'1970-01-01 00:00:01'` UTC to `'2038-01-19 03:14:07'` UTC. `TIMESTAMP` values are stored as the number of seconds since the epoch (`'1970-01-01 00:00:00'` UTC). A `TIMESTAMP` cannot represent the value `'1970-01-01 00:00:00'` because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing `'0000-00-00 00:00:00'`, the "zero" `TIMESTAMP` value.

  MySQL displays `TIMESTAMP` values in `'YYYY-MM-DD HH:MM:SS'` format. To convert the value to a number, add `+0`.

  A `TIMESTAMP` column is useful for recording the date and time of an `INSERT` or `UPDATE` operation. By default, the first `TIMESTAMP` column in a table is automatically set to the date and time of the most recent operation if you do not assign it a value yourself. You can also set any `TIMESTAMP` column to the current date and time by assigning it a `NULL` value. Variations on automatic initialization and update properties are described in Section 10.3.1.2, "`TIMESTAMP` Properties as of MySQL 4.1".

  In MySQL 4.1, `TIMESTAMP` is returned as a string with the format `'YYYY-MM-DD HH:MM:SS'`. Display widths (used as described in the following paragraphs) are no longer supported; the display width is fixed at 19 characters. To obtain the value as a number, add `+0`.

  In MySQL 4.0 and earlier, `TIMESTAMP` values are displayed in `YYYYMMDDHHMMSS`, `YYMMDDHHMMSS`, `YYYYMMDD`, or `YYMMDD` format, depending on whether *`M`* is 14 (or missing), 12, 8, or 6, but permits you to assign values to `TIMESTAMP` columns using either strings or numbers. The *`M`* argument affects only how a `TIMESTAMP` column is displayed, not storage. Its values always are stored using four bytes each. From MySQL 4.0.12, the `--new` option can be used to make the server behave as in MySQL 4.1.

  Note that `TIMESTAMP(M)` columns where *`M`* is 8 or 14 are reported to be numbers, whereas other `TIMESTAMP(M)` columns are reported to be strings. This is just to ensure that you can reliably dump and restore the table with these types.

  Note

  The behavior of `TIMESTAMP` columns changed considerably in MySQL 4.1. For complete information on the differences with regard to this data type in MySQL 4.1 and later versions (as opposed to MySQL 4.0 and earlier versions), be sure to see Section 10.3.1.1, "`TIMESTAMP` Properties Prior to MySQL 4.1", and Section 10.3.1.2, "`TIMESTAMP` Properties as of MySQL 4.1".

- `TIME`

  A time. The range is `'-838:59:59'` to `'838:59:59'`. MySQL displays `TIME` values in `'HH:MM:SS'` format, but permits assignment of values to `TIME` columns using either strings or numbers.

- `YEAR[(2|4)]`

  A year in two-digit or four-digit format. The default is four-digit format. In four-digit format, the permissible values are `1901` to `2155`, and `0000`. In two-digit format, the permissible values are `70` to `69`, representing years from 1970 to 2069. MySQL displays `YEAR` values in `YYYY` format, but permits assignment of values to `YEAR` columns using either strings or numbers. The `YEAR` type is unavailable prior to MySQL 3.22.

The SUM() [884] and AVG() [881] aggregate functions do not work with temporal values. (They convert the values to numbers, which loses the part after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

10.1.3 String Type Overview

A summary of the string data types follows. For additional information about properties and storage requirements of the string types, see Section 10.4, “String Types”, and Section 10.5, “Data Type Storage Requirements”.

In some cases, MySQL may change a string column to a type different from that given in a CREATE TABLE or ALTER TABLE statement. See Section 12.1.5.2, “Silent Column Specification Changes”.

In MySQL 4.1 and up, string data types include some features that you may not have encountered in working with versions of MySQL prior to 4.1:

- As of version 4.1, MySQL interprets length specifications in character column definitions in character units. (Before MySQL 4.1, column lengths were interpreted in bytes.) This applies to CHAR, VARCHAR, and the TEXT types.
- Column definitions for many string data types can include attributes that specify the character set or collation of the column. These attributes apply to the CHAR, VARCHAR, the TEXT types, ENUM, and SET data types:
  - The CHARACTER SET attribute specifies the character set, and the COLLATE attribute specifies a collation for the character set. For example:

    ```
    CREATE TABLE t
    (
        c1 VARCHAR(20) CHARACTER SET utf8,
        c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
    );
    ```

    This table definition creates a column named c1 that has a character set of utf8 with the default collation for that character set, and a column named c2 that has a character set of latin1 and a case-sensitive collation.

    The rules for assigning the character set and collation when either or both of the CHARACTER SET and COLLATE attributes are missing are described in Section 9.1.3.4, “Column Character Set and Collation”.

    CHARSET is a synonym for CHARACTER SET.
  - From MySQL 4.1.2 on, specifying the CHARACTER SET binary attribute for a character data type causes the column to be created as the corresponding binary data type: CHAR becomes BINARY, VARCHAR becomes VARBINARY, and TEXT becomes BLOB. For the ENUM and SET data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

    ```
    CREATE TABLE t
    (
      c1 VARCHAR(10) CHARACTER SET binary,
      c2 TEXT CHARACTER SET binary,
      c3 ENUM('a','b','c') CHARACTER SET binary
    ```

```
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

- From MySQL 4.1.0 on, the `ASCII` attribute is shorthand for `CHARACTER SET latin1`.
- From MySQL 4.1.1 on, the `UNICODE` attribute is shorthand for `CHARACTER SET ucs2`.
- As of MySQL 4.1.2, the `BINARY` attribute is shorthand for specifying the binary collation of the column character set. In this case, sorting and comparison are based on numeric character values. (Before MySQL 4.1.2, `BINARY` caused a column to store binary strings and sorting and comparison were based on numeric byte values. This is the same as using character values for single-byte character sets, but not for multi-byte character sets.)

- Character column sorting and comparison are based on the character set assigned to the column. (Before MySQL 4.1, sorting and comparison were based on the collation of the server character set.) For the `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET` data types, you can declare a column with a binary collation or the `BINARY` attribute to cause sorting and comparison to use the underlying character code values rather than a lexical ordering.

Section 9.1, “Character Set Support”, provides additional information about use of character sets in MySQL 4.1 and up.

- `[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

  A fixed-length string that is always right-padded with spaces to the specified length when stored. *`M`* represents the column length in characters. The range of *`M`* is 0 to 255. (1 to 255 prior to MySQL 3.23). If *`M`* is omitted, the length is 1.

  Note

  Trailing spaces are removed when `CHAR` values are retrieved.

  In MySQL 4.1, a `CHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `CHAR(500)` is converted to `TEXT`, and `CHAR(200000)` is converted to `MEDIUMTEXT`. This is a compatibility feature. However, this conversion causes the column to become a variable-length column, and also affects trailing-space removal.

  `CHAR` is shorthand for `CHARACTER`. `NATIONAL CHAR` (or its equivalent short form, `NCHAR`) is the standard SQL way to define that a `CHAR` column should use some predefined character set. MySQL 4.1 and up uses `utf8` as this predefined character set. Section 9.1.3.6, “National Character Set”.

  From MySQL 4.1.2 on, the `CHAR BYTE` data type is an alias for the `BINARY` data type. This is a compatibility feature.

  MySQL permits you to create a column of type `CHAR(0)`. This is useful primarily when you have to be compliant with old applications that depend on the existence of a column but that do not actually use its value. `CHAR(0)` is also quite nice when you need a column that can take only two values: A column

that is defined as `CHAR(0) NULL` occupies only one bit and can take only the values `NULL` and `''` (the empty string).

- `[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

  A variable-length string. *`M`* represents the maximum column length in characters. The range of *`M`* is 1 to 255 before MySQL 4.0.2, and 0 to 255 as of MySQL 4.0.2.

  MySQL stores `VARCHAR` values as a one-byte length prefix plus data. The length prefix indicates the number of bytes in the value.

  Note

  Trailing spaces are removed when `VARCHAR` values are stored. This differs from the standard SQL specification.

  In MySQL 4.1, a `VARCHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `VARCHAR(500)` is converted to `TEXT`, and `VARCHAR(200000)` is converted to `MEDIUMTEXT`. This is a compatibility feature. However, this conversion affects trailing-space removal.

  `VARCHAR` is shorthand for `CHARACTER VARYING`. `NATIONAL VARCHAR` is the standard SQL way to define that a `VARCHAR` column should use some predefined character set. MySQL 4.1 and up uses `utf8` as this predefined character set. Section 9.1.3.6, “National Character Set”. As of MySQL 4.1.1, `NVARCHAR` is shorthand for `NATIONAL VARCHAR`.

- `BINARY(M)`

  The `BINARY` type is similar to the `CHAR` type, but stores binary byte strings rather than nonbinary character strings. *`M`* represents the column length in bytes.

  This type was added in MySQL 4.1.2.

- `VARBINARY(M)`

  The `VARBINARY` type is similar to the `VARCHAR` type, but stores binary byte strings rather than nonbinary character strings. *`M`* represents the maximum column length in bytes.

  This type was added in MySQL 4.1.2.

- `TINYBLOB`

  A `BLOB` column with a maximum length of 255 (2^8 – 1) bytes. Each `TINYBLOB` value is stored using a one-byte length prefix that indicates the number of bytes in the value.

- `TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 255 (2^8 – 1) characters. The effective maximum length is less if the value contains multi-byte characters. Each `TINYTEXT` value is stored using a one-byte length prefix that indicates the number of bytes in the value.

- `BLOB[(M)]`

  A `BLOB` column with a maximum length of 65,535 (2^{16} – 1) bytes. Each `BLOB` value is stored using a two-byte length prefix that indicates the number of bytes in the value.

Beginning with MySQL 4.1, an optional length *M* can be given for this type. MySQL creates the column as the smallest BLOB type large enough to hold values *M* bytes long.

- TEXT[(*M*)] [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 65,535 (2^{16} – 1) characters. The effective maximum length is less if the value contains multi-byte characters. Each TEXT value is stored using a two-byte length prefix that indicates the number of bytes in the value.

  Beginning with MySQL 4.1, an optional length *M* can be given for this type. MySQL creates the column as the smallest TEXT type large enough to hold values *M* characters long.

- MEDIUMBLOB

  A BLOB column with a maximum length of 16,777,215 (2^{24} – 1) bytes. Each MEDIUMBLOB value is stored using a three-byte length prefix that indicates the number of bytes in the value.

- MEDIUMTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 16,777,215 (2^{24} – 1) characters. The effective maximum length is less if the value contains multi-byte characters. Each MEDIUMTEXT value is stored using a three-byte length prefix that indicates the number of bytes in the value.

- LONGBLOB

  A BLOB column with a maximum length of 4,294,967,295 or 4GB (2^{32} – 1) bytes. Up to MySQL 3.23, the client/server protocol and MyISAM tables had a limit of 16MB per communication packet or table row. As of MySQL 4.0, the effective maximum length of LONGBLOB columns depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGBLOB value is stored using a four-byte length prefix that indicates the number of bytes in the value.

- LONGTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 4,294,967,295 or 4GB (2^{32} – 1) characters. The effective maximum length is less if the value contains multi-byte characters. Up to MySQL 3.23, the client/server protocol and MyISAM tables had a limit of 16MB per communication packet or table row. As of MySQL 4.0, the effective maximum length of LONGTEXT columns depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGTEXT value is stored using a four-byte length prefix that indicates the number of bytes in the value.

- ENUM('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  An enumeration. A string object that can have only one value, chosen from the list of values '*value1*', '*value2*', ..., NULL or the special '' error value. An ENUM column can have a maximum of 65,535 distinct values. ENUM values are represented internally as integers.

- SET('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A set. A string object that can have zero or more values, each of which must be chosen from the list of values '*value1*', '*value2*', ... A SET column can have a maximum of 64 members. SET values are represented internally as integers.

10.1.4 Data Type Default Values

The DEFAULT *value* clause in a data type specification indicates a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as NOW() [837] or CURRENT_DATE [829]. The exception is that you can specify CURRENT_TIMESTAMP [829] as the default for a TIMESTAMP column as of MySQL 4.1.2. See Section 10.3.1.2, "TIMESTAMP Properties as of MySQL 4.1".

If a column definition includes no explicit DEFAULT value, MySQL determines the default value as follows:

If the column can take NULL as a value, the column is defined with an explicit DEFAULT NULL clause.

If the column cannot take NULL as the value, MySQL defines the column with an explicit DEFAULT clause, using the implicit default value for the column data type. Implicit defaults are defined as follows:

- For numeric types, the default is 0, with the exception that for integer or floating-point types declared with the AUTO_INCREMENT attribute, the default is the next value in the sequence.
- For date and time types other than TIMESTAMP, the default is the appropriate "zero" value for the type. For the first TIMESTAMP column in a table, the default value is the current date and time. See Section 10.3, "Date and Time Types".
- For string types other than ENUM, the default value is the empty string. For ENUM, the default is the first enumeration value.

BLOB and TEXT columns cannot be assigned a default value.

For a given table, you can use the SHOW CREATE TABLE statement to see which columns have an explicit DEFAULT clause.

SERIAL DEFAULT VALUE in the definition of an integer column is an alias for NOT NULL AUTO_INCREMENT UNIQUE.

10.2 Numeric Types

MySQL supports all the standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION). The keyword INT is a synonym for INTEGER, and the keyword DEC is a synonym for DECIMAL. MySQL treats DOUBLE as a synonym for DOUBLE PRECISION (a nonstandard extension). MySQL also treats REAL as a synonym for DOUBLE PRECISION (a nonstandard variation), unless the REAL_AS_FLOAT [460] SQL mode is enabled.

For information about how MySQL handles assignment of out-of-range values to columns and overflow during expression evaluation, see Section 10.2.5, "Out-of-Range and Overflow Handling".

For information about numeric type storage requirements, see Section 10.5, "Data Type Storage Requirements".

The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them. For more information, see Section 11.6.1, "Arithmetic Operators".

10.2.1 Integer Types (Exact Value)

MySQL supports the SQL standard integer types INTEGER (or INT) and SMALLINT. As an extension to the standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT. The following table shows the required storage and range for each integer type.

Type	Storage	Minimum Value	Maximum Value
	(Bytes)	(Signed/Unsigned)	Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

10.2.2 Fixed-Point Types (Exact Value)

The DECIMAL and NUMERIC types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, NUMERIC is implemented as DECIMAL, so the following remarks about DECIMAL apply equally to NUMERIC.

Through version 4.1, MySQL stores DECIMAL values as strings, rather than in binary format. One character is used for each digit of the value, the decimal point (if the scale is greater than 0), and the "-" sign (for negative numbers).

In a DECIMAL column declaration, the precision and scale can be (and usually is) specified; for example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

Standard SQL requires that DECIMAL(5,2) be able to store any value with five digits and two decimals, so values that can be stored in the salary column range from -999.99 to 999.99. In versions up to and including 4.1, MySQL varies from this limit in two ways due to the use of string format for value storage:

- On the positive end of the range, the column actually can store numbers up to 9999.99. For positive numbers, MySQL uses the byte reserved for the sign to extend the upper end of the range.
- DECIMAL columns in MySQL before 3.23 are stored differently and cannot represent all the values required by standard SQL. This is because for a type of DECIMAL(*M*,*D*), the value of *M* includes the bytes for the sign and the decimal point. The range of the salary column before MySQL 3.23 would be -9.99 to 99.99.

In standard SQL, the syntax DECIMAL(*M*) is equivalent to DECIMAL(*M*,0). Similarly, the syntax DECIMAL is equivalent to DECIMAL(*M*,0), where the implementation is permitted to decide the value of *M*. As of MySQL 3.23.6, both of these variant forms of DECIMAL syntax are supported. The default value of *M* is 10. Before 3.23.6, *M* and *D* both must be specified explicitly.

If the scale is 0, DECIMAL values contain no decimal point or fractional part.

The maximum range of DECIMAL values is the same as for DOUBLE, but the actual range for a given DECIMAL column can be constrained by the precision or scale for a given column. When such a column is

assigned a value with more digits following the decimal point than are permitted by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the permissible number of digits.)

10.2.3 Floating-Point Types (Approximate Value)

The `FLOAT` and `DOUBLE` types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

For `FLOAT`, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword `FLOAT` in parentheses. MySQL also supports this optional precision specification, but the precision value is used only to determine storage size. A precision from 0 to 23 results in a four-byte single-precision `FLOAT` column. A precision from 24 to 53 results in an eight-byte double-precision `DOUBLE` column.

MySQL permits a nonstandard syntax: `FLOAT(M,D)` or `REAL(M,D)` or `DOUBLE PRECISION(M,D)`. Here, "`(M,D)`" means than values can be stored with up to `M` digits in total, of which `D` digits may be after the decimal point. For example, a column defined as `FLOAT(7,4)` will look like `-999.9999` when displayed. MySQL performs rounding when storing values, so if you insert `999.00009` into a `FLOAT(7,4)` column, the approximate result is `999.0001`.

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. For more information, see Section B.5.5.8, "Problems with Floating-Point Values"

For maximum portability, code requiring storage of approximate numeric data values should use `FLOAT` or `DOUBLE PRECISION` with no specification of precision or number of digits.

10.2.4 Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, `INT(4)` specifies an `INT` with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used or not is up to the application.)

The display width does *not* constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as `SMALLINT(3)` has the usual `SMALLINT` range of `-32768` to `32767`, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) attribute `ZEROFILL`, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(4) ZEROFILL`, a value of `5` is retrieved as `0005`.

Note

The `ZEROFILL` attribute is ignored when a column is involved in expressions or `UNION` queries.

If you store values larger than the display width in an integer column that has the `ZEROFILL` attribute, you may experience problems when MySQL generates temporary tables for some complicated joins. In these cases, MySQL assumes that the data values fit within the column display width.

All integer types can have an optional (nonstandard) attribute `UNSIGNED`. Unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the

column. For example, if an `INT` column is `UNSIGNED`, the size of the column's range is the same but its endpoints shift from `-2147483648` and `2147483647` up to `0` and `4294967295`.

As of MySQL 4.0.2, floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Integer or floating-point data types can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is *`value`*`+1`, where *`value`* is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`.

10.2.5 Out-of-Range and Overflow Handling

When MySQL stores a value in a numeric column that is outside the permissible range of the column data type, MySQL clips the value to the appropriate endpoint of the range and stores the resulting value instead.

For example, when an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range. If you store 256 into a `TINYINT` or `TINYINT UNSIGNED` column, MySQL stores 127 or 255, respectively. When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Column-assignment conversions that occur due to clipping are reported as warnings for `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, and multiple-row `INSERT` statements.

In MySQL 4.1, overflow handling during numeric expression evaluation depends on the types of the operands:

- Integer overflow results in silent wraparound.
- `DECIMAL` overflow results in a truncated result and a warning.
- Floating-point overflow produces a `NULL` result. Overflow for some operations can result in `+INF`, `-INF`, or `NaN`.

For example, the largest signed `BIGINT` value is 9223372036854775807, so the following expression wraps around to the minimum `BIGINT` value:

```
mysql> SELECT 9223372036854775807 + 1;
+--------------------------+
| 9223372036854775807 + 1 |
+--------------------------+
|    -9223372036854775808 |
+--------------------------+
```

To enable the operation to succeed in this case, convert the value to unsigned;

```
mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;
+-----------------------------------------+
| CAST(9223372036854775807 AS UNSIGNED) + 1 |
+-----------------------------------------+
|                      9223372036854775808 |
+-----------------------------------------+
```

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because DECIMAL values have a larger range than integers:

```
mysql> SELECT 9223372036854775807.0 + 1;
+--------------------------+
| 9223372036854775807.0 + 1 |
+--------------------------+
|     9223372036854775808.0 |
+--------------------------+
```

Subtraction between integer values, where one is of type UNSIGNED, produces an unsigned result by default. If the result would otherwise have been negative, it becomes the maximum integer value. If the NO_UNSIGNED_SUBTRACTION [459] SQL mode is enabled, the result is negative.

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|    18446744073709551615 |
+-------------------------+

mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|                      -1 |
+-------------------------+
```

If the result of such an operation is used to update an UNSIGNED integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if NO_UNSIGNED_SUBTRACTION [459] is enabled.

10.3 Date and Time Types

The date and time types for representing temporal values are DATE, TIME, DATETIME, TIMESTAMP, and YEAR. Each temporal type has a range of legal values, as well as a “zero” value that is used when you specify an illegal value that MySQL cannot represent. The TIMESTAMP type has special automatic updating behavior, described later on. For temporal type storage requirements, see Section 10.5, “Data Type Storage Requirements”.

MySQL versions through 4.1 accept certain “illegal” values for dates, such as '2009-11-31'. This is useful when you want to store a possibly incorrect value specified by a user (for example, in a web form) in the database for future processing. MySQL verifies only that the month is in the range from 0 to 12 and that the day is in the range from 0 to 31. These ranges are defined to include zero because MySQL permits you to store dates where the day or month and day are zero in a DATE or DATETIME column. This is extremely useful for applications that need to store a birthdate for which you do not know the exact date. In this case, you simply store the date as '2009-00-00' or '2009-01-00'. If you store dates such as these, you should not expect to get correct results for functions such as DATE_SUB() [833] or DATE_ADD() [829] that require complete dates.

MySQL also permits you to store '0000-00-00' as a “dummy date.” This is in some cases more convenient (and uses less data and index space) than storing NULL values.

Keep in mind these general considerations when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). For a description of the permitted formats for date

and time types, see Section 8.1.3, “Date and Time Literals”. It is expected that you supply legal values. Unpredictable results may occur if you use values in other formats.

- Although MySQL tries to interpret values in several formats, date parts must always be given in year-month-day order (for example, `'98-09-04'`), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, `'09-04-98'`, `'04-09-98'`).
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:
  - Year values in the range `70-99` are converted to `1970-1999`.
  - Year values in the range `00-69` are converted to `2000-2069`.

  See also Section 10.3.6, “Two-Digit Years in Dates”.
- Conversion of values from one temporal type to another occurs according to the rules in Section 10.3.5, “Conversion Between Date and Time Types”.
- MySQL automatically converts a date or time value to a number if the value is used in a numeric context and vice versa.
- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise illegal for the type, it converts the value to the “zero” value for that type. The exception is that out-of-range `TIME` values are clipped to the appropriate endpoint of the `TIME` range.
- “Zero” date or time values used through MyODBC are converted automatically to `NULL` in MyODBC 2.50.12 and above, because ODBC cannot handle such values.

The following table shows the format of the “zero” value for each type. The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values `'0'` or `0`, which are easier to write.

Data Type	“Zero” Value
`DATE`	`'0000-00-00'`
`TIME`	`'00:00:00'`
`DATETIME`	`'0000-00-00 00:00:00'`
`TIMESTAMP` (4.1 and up)	`'0000-00-00 00:00:00'`
`TIMESTAMP` (before 4.1)	`00000000000000`
`YEAR`	`0000`

10.3.1 The DATE, DATETIME, and TIMESTAMP Types

The `DATE`, `DATETIME`, and `TIMESTAMP` types are related. This section describes their characteristics, how they are similar, and how they differ.

The `DATE` type is used for values with a date part but no time part. MySQL retrieves and displays `DATE` values in `'YYYY-MM-DD'` format. The supported range is `'1000-01-01'` to `'9999-12-31'`.

The `DATETIME` type is used for values that contain both date and time parts. MySQL retrieves and displays `DATETIME` values in `'YYYY-MM-DD HH:MM:SS'` format. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`.

For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The TIMESTAMP data type is used for values that contain both date and time parts. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC. Its properties are described in more detail in Section 10.3.1.2, "TIMESTAMP Properties as of MySQL 4.1".

MySQL recognizes DATE, DATETIME, and TIMESTAMP values in several formats, described in Section 8.1.3, "Date and Time Literals". A DATETIME or TIMESTAMP value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into DATETIME or TIMESTAMP columns. For information about fractional seconds support in MySQL, see Section 10.3.4, "Fractional Seconds in Time Values".

Illegal DATE, DATETIME, or TIMESTAMP values are converted to the "zero" value of the appropriate type ('0000-00-00', '0000-00-00 00:00:00', or 00000000000000).

Be aware of certain problems when specifying date values:

- MySQL permits a "relaxed" format for values specified as strings, in which any punctuation character may be used as the delimiter between date parts or time parts. In some cases, this syntax can be deceiving. For example, a value such as '10:11:12' might look like a time value because of the ":" delimiter, but is interpreted as the year '2010-11-12' if used in a date context. The value '10:45:15' is converted to '0000-00-00' because '45' is not a legal month.

- The MySQL server performs only basic checking on the validity of a date: The ranges for year, month, and day are 1000 to 9999, 00 to 12, and 00 to 31, respectively. Any date containing parts not within these ranges is subject to conversion to '0000-00-00'. Please note that this still permits you to store invalid dates such as '2002-04-31'. To ensure that a date is valid, perform a check in your application.

- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

  - Year values in the range 00-69 are converted to 2000-2069.

  - Year values in the range 70-99 are converted to 1970-1999.

  See also Section 10.3.6, "Two-Digit Years in Dates".

10.3.1.1 TIMESTAMP Properties Prior to MySQL 4.1

TIMESTAMP values are converted from the current time zone to UTC for storage, and converted back from UTC to the current time zone for retrieval. (This occurs only for the TIMESTAMP data type, not for other types such as DATETIME.)

The TIMESTAMP data type provides a type that you can use to automatically mark INSERT or UPDATE operations with the current date and time. If you have multiple TIMESTAMP columns in a table, only the first one is updated automatically. (From MySQL 4.1.2 on, you can specify which TIMESTAMP column updates; see Section 10.3.1.2, "TIMESTAMP Properties as of MySQL 4.1".)

Automatic updating of the first TIMESTAMP column in a table occurs under any of the following conditions:

- You explicitly set the column to NULL.

- The column is not specified explicitly in an INSERT or LOAD DATA INFILE statement.

- The column is not specified explicitly in an UPDATE statement and some other column changes value. An UPDATE that sets a column to the value it does not cause the TIMESTAMP column to be updated; if you set a column to its current value, MySQL ignores the update for efficiency.

A TIMESTAMP column other than the first also can be assigned the current date and time by setting it to NULL or to any function that produces the current date and time (NOW() [837], CURRENT_TIMESTAMP [829]).

Note that the information in the following discussion applies to TIMESTAMP columns only for tables not created with MAXDB [460] mode enabled, because such columns are created as DATETIME columns.

You can set any TIMESTAMP column to a value different from the current date and time by setting it explicitly to the desired value. This is true even for the first TIMESTAMP column. You can use this property if, for example, you want a TIMESTAMP to be set to the current date and time when you create a row, but not to be changed whenever the row is updated later:

- Let MySQL set the column when the row is created. This initializes it to the current date and time.
- When you perform subsequent updates to other columns in the row, set the TIMESTAMP column explicitly to its current value:

```
UPDATE tbl_name
    SET timestamp_col = timestamp_col,
        other_col1 = new_value1,
        other_col2 = new_value2, ...
```

Another way to maintain a column that records row-creation time is to use a DATETIME column that you initialize to NOW() [837] when the row is created and do not modify for subsequent updates.

TIMESTAMP values may range from the beginning of 1970 to partway through the year 2038, with a resolution of one second. Values are displayed as numbers. When you store a value in a TIMESTAMP column, it is assumed to be represented in the current time zone, and is converted to UTC for storage. When you retrieve the value, it is converted from UTC back to the local time zone for display. Before MySQL 4.1.3, the server has a single time zone. As of 4.1.3, clients can set their own time zones on a per-connection basis, as described in Section 9.7, “MySQL Server Time Zone Support”.

Prior to version 4.1, the format in which MySQL retrieves and displays TIMESTAMP values depends on the display size, as illustrated in the following table. The “full” TIMESTAMP format is 14 digits, but TIMESTAMP columns may be created with shorter display sizes.

Data Type	Display Format
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

All TIMESTAMP columns have the same storage size, regardless of display size. The most common display sizes are 6, 8, 12, and 14. You can specify an arbitrary display size at table creation time, but values of 0 or greater than 14 are coerced to 14. Odd-valued sizes in the range from 1 to 13 are coerced to the next higher even number.

TIMESTAMP columns store legal values using the full precision with which the value was specified, regardless of the display size. This has several implications:

- Always specify year, month, and day, even if your column types are TIMESTAMP(4) or TIMESTAMP(2). Otherwise, the value is not a legal date and 0 is stored.

- If you use ALTER TABLE to widen a narrow TIMESTAMP column, information is displayed that previously was “hidden.”
- Similarly, narrowing a TIMESTAMP column does not cause information to be lost, except in the sense that less information is shown when the values are displayed.
- If you are planning to use mysqldump for the database, do not use TIMESTAMP(4) or TIMESTAMP(2). The display format for these data types are not legal dates and 0 will be stored instead. This inconsistency is fixed starting with MySQL 4.1, where display width is ignored. To prepare for transition to versions after 4.0, you should change to use display widths of 6 or more, which will produce a legal display format. You can change the display width of TIMESTAMP data types, without losing any information, by using ALTER TABLE as indicated above.

  If you need to print the timestamps for external applications, you can use MID() [799] to extract the relevant part of the timestamp: for example, to imitate the TIMESTAMP(4) display format.
- Although TIMESTAMP values are stored to full precision, the only function that operates directly on the underlying stored value is UNIX_TIMESTAMP() [841]. Other functions operate on the formatted retrieved value. This means you cannot use a function such as HOUR() [836] or SECOND() [838] unless the relevant part of the TIMESTAMP value is included in the formatted value. For example, the HH part of a TIMESTAMP column is not displayed unless the display size is at least 10, so trying to use HOUR() [836] on shorter TIMESTAMP values produces a meaningless result.

In MySQL 4.1, TIMESTAMP display format changes to be the same as DATETIME, that is, as a string in 'YYYY-MM-DD HH:MM:SS' format rather than as a number in YYYYMMDDHHMMSS format. To test applications written for MySQL 4.0 for compatibility with this change, you can set the new [390] system variable to 1. This variable is available beginning with MySQL 4.0.12. It can be set at server startup by specifying the --new option to mysqld. At runtime, a user who has the SUPER [493] privilege can set the global value with a SET statement:

```
mysql> SET GLOBAL new = 1;
```

Any client can set its session value of new [390] as follows:

```
mysql> SET new = 1;
```

The general effect of setting new [390] to 1 is that values for existing TIMESTAMP columns display as strings rather than as numbers. Also, DESCRIBE displays the column definition as TIMESTAMP(19), rather than as TIMESTAMP(14).

The effect differs somewhat for TIMESTAMP columns that are created while new [390] is set to 1. In this case, column values display as strings and DESCRIBE shows the definition as TIMESTAMP(19), regardless of the current value of new [390].

In other words, with new=1, all TIMESTAMP values display as strings and DESCRIBE shows a display width of 19. For columns created while new=1, they continue to display as strings and to have a display width of 19 even if new [390] is set to 0.

For a TIMESTAMP column that displays as a string, you can display it as a number by retrieving it as *col_name*+0.

10.3.1.2 TIMESTAMP Properties as of MySQL 4.1

In MySQL 4.1 and up, the properties of the TIMESTAMP data type changed in several ways. The following discussion describes the revised syntax and behavior.

Beginning with MySQL 4.1.3, the default current time zone for each connection is the server's time. The time zone can be set on a per-connection basis, as described in Section 9.7, “MySQL Server Time Zone Support”. TIMESTAMP values still are stored in UTC, but are converted from the current time zone for storage, and converted back to the current time zone for retrieval. As long as the time zone setting remains constant, you get back the same value you store. If you store a TIMESTAMP value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the time_zone [431] system variable.

From MySQL 4.1.0 on, TIMESTAMP display format differs from that of earlier MySQL releases:

- TIMESTAMP columns are displayed in the same format as DATETIME columns. In other words, the display width is fixed at 19 characters, and the format is 'YYYY-MM-DD HH:MM:SS'.
- Display widths (used as described in the preceding section) are no longer supported. In other words, for declarations such as TIMESTAMP(2), TIMESTAMP(4), and so on, the display width is ignored.

The following items summarize TIMESTAMP initialization and updating properties prior to MySQL 4.1.2:

- The first TIMESTAMP column in table row automatically is set to the current timestamp when the record is created if the column is set to NULL or is not specified at all.
- The first TIMESTAMP column in table row automatically is updated to the current timestamp when the value of any other column in the row is changed, unless the TIMESTAMP column explicitly is assigned a value other than NULL.
- If a DEFAULT value is specified for the first TIMESTAMP column when the table is created, it is silently ignored.
- Other TIMESTAMP columns in the table can be set to the current TIMESTAMP by assigning NULL to them, but they do not update automatically.

Beginning with MySQL 4.1.2, you have more flexible control over when automatic TIMESTAMP initialization and updating occur and which column should have those behaviors:

- For one TIMESTAMP column in a table, you can assign the current timestamp as the default value and the auto-update value. It is possible to have the current timestamp be the default value for initializing the column, for the auto-update value, or both. It is not possible to have the current timestamp be the default value for one column and the auto-update value for another column.
- Any single TIMESTAMP column in a table can be used as the one that is initialized to the current date and time, or updated automatically. This need not be the first TIMESTAMP column.
- In a CREATE TABLE statement, the first TIMESTAMP column can be declared in any of the following ways:
  - With both DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP clauses, the column has the current timestamp for its default value, and is automatically updated.
  - With neither DEFAULT nor ON UPDATE clauses, it is the same as DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP.
  - With a DEFAULT CURRENT_TIMESTAMP clause and no ON UPDATE clause, the column has the current timestamp for its default value but is not automatically updated.
  - With no DEFAULT clause and with an ON UPDATE CURRENT_TIMESTAMP clause, the column has a default of 0 and is automatically updated.

- With a constant DEFAULT value, the column has the given default and is not automatically initialized to the current timestamp. If the column also has an ON UPDATE CURRENT_TIMESTAMP clause, it is automatically updated; otherwise, it has a constant default and is not automatically updated.

In other words, you can use the current timestamp for both the initial value and the auto-update value, or either one, or neither. (For example, you can specify ON UPDATE to enable auto-update without also having the column auto-initialized.) The following column definitions demonstrate each possibility:

- Auto-initialization and auto-update:

```
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

- Auto-initialization only:

```
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

- Auto-update only:

```
ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
```

- Neither:

```
ts TIMESTAMP DEFAULT 0
```

- To specify automatic default or updating for a TIMESTAMP column other than the first one, you must suppress the automatic initialization and update behaviors for the first TIMESTAMP column by explicitly assigning it a constant DEFAULT value (for example, DEFAULT 0 or DEFAULT '2003-01-01 00:00:00'). Then, for the other TIMESTAMP column, the rules are the same as for the first TIMESTAMP column, except that if you omit both of the DEFAULT and ON UPDATE clauses, no automatic initialization or updating occurs.

  Example:

```
CREATE TABLE t (
    ts1 TIMESTAMP DEFAULT 0,
    ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                  ON UPDATE CURRENT_TIMESTAMP);
```

- CURRENT_TIMESTAMP [829] or any of its synonyms (CURRENT_TIMESTAMP() [829], NOW() [837], LOCALTIME [836], LOCALTIME() [836], LOCALTIMESTAMP [836], or LOCALTIMESTAMP() [836]) can be used in the DEFAULT and ON UPDATE clauses. They all mean “the current timestamp.” UTC_TIMESTAMP [843] is not permitted. Its range of values does not align with those of the TIMESTAMP column except when the current time zone is UTC.

- The order of the DEFAULT and ON UPDATE clauses does not matter. If both DEFAULT and ON UPDATE are specified for a TIMESTAMP column, either can precede the other. For example, these statements are equivalent:

```
CREATE TABLE t (ts TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                             ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
                             DEFAULT CURRENT_TIMESTAMP);
```

The following rules describe the changes in MySQL 4.1 regarding TIMESTAMP and handling of NULL values:

- Before MySQL 4.1.2, TIMESTAMP columns are NOT NULL. They cannot contain NULL values, and assigning NULL assigns the current timestamp. Any DEFAULT clause is ignored.
- From MySQL 4.1.2 to 4.1.5, TIMESTAMP columns are NOT NULL. They cannot contain NULL values, and assigning NULL assigns the current timestamp. A DEFAULT NULL clause can be specified, but it is treated as DEFAULT CURRENT_TIMESTAMP for the first TIMESTAMP column and as DEFAULT 0 for other TIMESTAMP columns.
- As of MySQL 4.1.6, TIMESTAMP columns are NOT NULL by default, cannot contain NULL values, and assigning NULL assigns the current timestamp. However, a TIMESTAMP column can be permitted to contain NULL by declaring it with the NULL attribute. In this case, the default value also becomes NULL unless overridden with a DEFAULT clause that specifies a different default value. DEFAULT NULL can be used to explicitly specify NULL as the default value. (For a TIMESTAMP column not declared with the NULL attribute, DEFAULT NULL is illegal.) If a TIMESTAMP column permits NULL values, assigning NULL sets it to NULL, not to the current timestamp.

The following table contains several TIMESTAMP columns that permit NULL values:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

Note that a TIMESTAMP column that permits NULL values will *not* take on the current timestamp except under one of the following conditions:

- Its default value is defined as CURRENT_TIMESTAMP [829]
- NOW() [837] or CURRENT_TIMESTAMP [829] is inserted into the column

In other words, a TIMESTAMP column defined as NULL will auto-initialize only if it is created using a definition such as the following:

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

Otherwise—that is, if the TIMESTAMP column is defined to permit NULL values but not using DEFAULT CURRENT_TIMESTAMP, as shown here…

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT NULL);
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
```

…then you must explicitly insert a value corresponding to the current date and time. For example:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

Note

Beginning with MySQL 4.1.1, the MySQL server can be run with the MAXDB [460] SQL mode enabled. When the server runs with this mode enabled, TIMESTAMP is identical with DATETIME. That is, if this mode is enabled at the time that a table is

> created, TIMESTAMP columns are created as DATETIME columns. As a result, such columns use DATETIME display format, have the same range of values, and there is no automatic initialization or updating to the current date and time.

To enable MAXDB [460] mode, set the server SQL mode to MAXDB [460] at startup using the --sql-mode=MAXDB [394] server option or by setting the global sql_mode [429] variable at runtime:

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

A client can cause the server to run in MAXDB [460] mode for its own connection as follows:

```
mysql> SET SESSION sql_mode=MAXDB;
```

10.3.2 The TIME Type

MySQL retrieves and displays TIME values in 'HH:MM:SS' format (or 'HHH:MM:SS' format for large hours values). TIME values may range from '-838:59:59' to '838:59:59'. The hours part may be so large because the TIME type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

MySQL recognizes TIME values in several formats, described in Section 8.1.3, “Date and Time Literals”. Some of these formats can include a trailing fractional seconds part in up to microseconds (6 digits) precision. Although this fractional part is recognized, it is discarded from values stored into TIME columns. For information about fractional seconds support in MySQL, see Section 10.3.4, “Fractional Seconds in Time Values”.

Be careful about assigning abbreviated values to a TIME column. MySQL interprets abbreviated TIME values with colons as time of the day. That is, '11:12' means '11:12:00', not '00:11:12'. MySQL interprets abbreviated values without colons using the assumption that the two rightmost digits represent seconds (that is, as elapsed time rather than as time of day). For example, you might think of '1112' and 1112 as meaning '11:12:00' (12 minutes after 11 o'clock), but MySQL interprets them as '00:11:12' (11 minutes, 12 seconds). Similarly, '12' and 12 are interpreted as '00:00:12'.

By default, values that lie outside the TIME range but are otherwise legal are clipped to the closest endpoint of the range. For example, '-850:00:00' and '850:00:00' are converted to '-838:59:59' and '838:59:59'. Illegal TIME values are converted to '00:00:00'. Note that because '00:00:00' is itself a legal TIME value, there is no way to tell, from a value of '00:00:00' stored in a table, whether the original value was specified as '00:00:00' or whether it was illegal.

For more restrictive treatment of invalid TIME values, enable strict SQL mode to cause errors to occur. See Section 5.1.6, “Server SQL Modes”.

10.3.3 The YEAR Type

The YEAR type is a one-byte type used for representing years. It can be declared as YEAR(2) or YEAR(4) to specify a display width of two or four characters. The default is four characters if no width is given.

For four-digit format, MySQL displays YEAR values in YYYY format, with a range of 1901 to 2155, or 0000. For two-digit format, MySQL displays only the last two (least significant) digits; for example, 70 (1970 or 2070) or 69 (2069).

You can specify input YEAR values in a variety of formats:

- As a four-digit string in the range '1901' to '2155'.

- As a four-digit number in the range `1901` to `2155`.
- As a two-digit string in the range `'00'` to `'99'`. Values in the ranges `'00'` to `'69'` and `'70'` to `'99'` are converted to `YEAR` values in the ranges `2000` to `2069` and `1970` to `1999`.
- As a two-digit number in the range `1` to `99`. Values in the ranges `1` to `69` and `70` to `99` are converted to `YEAR` values in the ranges `2001` to `2069` and `1970` to `1999`. Note that the range for two-digit numbers is slightly different from the range for two-digit strings, because you cannot specify zero directly as a number and have it be interpreted as `2000`. You must specify it as a string `'0'` or `'00'` or it is interpreted as `0000`.
- As the result of a function that returns a value that is acceptable in a `YEAR` context, such as `NOW()` [837].

Illegal `YEAR` values are converted to `0000`.

See also Section 10.3.6, “Two-Digit Years in Dates”.

10.3.4 Fractional Seconds in Time Values

A trailing fractional seconds part is permissible for temporal values in contexts such as literal values, and in the arguments to or return values from some temporal functions. Example:

```
mysql> SELECT MICROSECOND('2010-12-10 14:12:09.019473');
+-----------------------------------------+
| MICROSECOND('2010-12-10 14:12:09.019473') |
+-----------------------------------------+
|                                   19473 |
+-----------------------------------------+
```

However, when MySQL stores a value into a column of any temporal data type, it discards any fractional part and does not store it.

10.3.5 Conversion Between Date and Time Types

To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information. In all cases, conversion between temporal types is subject to the range of legal values for the resulting type. For example, although `DATE`, `DATETIME`, and `TIMESTAMP` values all can be specified using the same set of formats, the types do not all have the same range of values. `TIMESTAMP` values cannot be earlier than `1970` UTC or later than `'2038-01-19 03:14:07'` UTC. This means that a date such as `'1968-01-01'`, while legal as a `DATE` or `DATETIME` value, is not valid as a `TIMESTAMP` value and is converted to `0`.

Conversion of `DATE` values:

- Conversion to a `DATETIME` or `TIMESTAMP` value adds a time part of `'00:00:00'` because the `DATE` value contains no time information.
- Conversion to a `TIME` value is not useful; the result is `'00:00:00'`.

Conversion of `DATETIME` and `TIMESTAMP` values:

- Conversion to a `DATE` value discards the time part because the `DATE` type contains no time information.
- Conversion to a `TIME` value discards the date part because the `TIME` type contains no date information.

Conversion of `TIME` values:

MySQL converts a time value to a date or date-and-time value by parsing the string value of the time as a date or date-and-time. This is unlikely to be useful. For example, `'23:12:31'` interpreted as a date becomes `'2032-12-31'`. Time values not valid as dates become `'0000-00-00'` or `NULL`.

As of MySQL 4.1.13, conversion of `TIME` or `DATETIME` values to numeric form (for example, by adding `+0`) results in a double-precision value with a microseconds part of `.000000`:

```
mysql> SELECT CURTIME(), CURTIME()+0;
+-----------+---------------+
| CURTIME() | CURTIME()+0   |
+-----------+---------------+
| 10:41:36  | 104136.000000 |
+-----------+---------------+
mysql> SELECT NOW(), NOW()+0;
+---------------------+-----------------------+
| NOW()               | NOW()+0               |
+---------------------+-----------------------+
| 2007-11-30 10:41:47 | 20071130104147.000000 |
+---------------------+-----------------------+
```

Before MySQL 4.1.13, the conversion results in an integer value with no microseconds part.

10.3.6 Two-Digit Years in Dates

Date values with two-digit years are ambiguous because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For `DATETIME`, `DATE`, `TIMESTAMP`, and `YEAR` types, MySQL interprets dates specified with ambiguous year values using these rules:

- Year values in the range `00-69` are converted to `2000-2069`.
- Year values in the range `70-99` are converted to `1970-1999`.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the values you require, you must provide unambiguous input containing four-digit year values.

In MySQL, the `YEAR` data type can store the years `0` and `1901` to `2155` in one byte and display them using two or four digits. All two-digit years are considered to be in the range `1970` to `2069`, which means that if you store `01` in a `YEAR` column, MySQL Server treats it as `2001`.

`ORDER BY` properly sorts `YEAR` values that have two-digit years.

Some functions like `MIN()` [884] and `MAX()` [884] convert a `YEAR` to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the `YEAR` to four-digit year format.

10.4 String Types

The string types are `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, and `SET`. This section describes how these types work and how to use them in your queries. For string type storage requirements, see Section 10.5, “Data Type Storage Requirements”.

10.4.1 The `CHAR` and `VARCHAR` Types

The `CHAR` and `VARCHAR` types are similar, but differ in the way they are stored and retrieved.

The CHAR and VARCHAR types are declared with a length that indicates the maximum number of characters you want to store. For example, CHAR(30) can hold up to 30 characters. (Before MySQL 4.1, the length is interpreted as number of bytes.)

The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. (Before MySQL 3.23, the length of CHAR may be from 1 to 255.) When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed.

Values in VARCHAR columns are variable-length strings. The length can be specified as a value from 1 to 255 before MySQL 4.0.2 and 0 to 255 as of MySQL 4.0.2.

In contrast to CHAR, VARCHAR values are stored as a one-byte length prefix plus data. The length prefix indicates the number of bytes in the value.

If you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit. If the truncated characters are not spaces, a warning is generated.

VARCHAR values are not padded when they are stored. Trailing spaces in MySQL version up to and including 4.1 are removed from values when stored in a VARCHAR column; this also means that the spaces are absent from retrieved values.

If you need a data type for which trailing spaces are not removed, consider using a BLOB or TEXT type. If you want to store binary values such as results from an encryption or compression function that might contain arbitrary byte values, use a BLOB column rather than a CHAR or VARCHAR column, to avoid potential problems with trailing space removal that would change data values.

The following table illustrates the differences between CHAR and VARCHAR by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns (assuming that the column uses a single-byte character set such as latin1).

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

If a given value is stored into the CHAR(4) and VARCHAR(4) columns, the values retrieved from the columns are not always the same because trailing spaces are removed from CHAR columns upon retrieval.

As of MySQL 4.1, values in CHAR and VARCHAR columns are sorted and compared according to the character set collation assigned to the column. Before MySQL 4.1, sorting and comparison are based on the collation of the server character set; you can declare the column with the BINARY attribute to cause sorting and comparison to be based on the numeric values of the bytes in column values. BINARY does not affect how column values are stored or retrieved.

All MySQL collations are of type PADSPACE. This means that all CHAR and VARCHAR values in MySQL are compared without regard to any trailing spaces. For example:

```
mysql> CREATE TABLE names (myname CHAR(10), yourname VARCHAR(10));
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO names VALUES ('Monty ', 'Monty ');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT myname = 'Monty  ', yourname = 'Monty  ' FROM names;
+--------------------+----------------------+
| myname = 'Monty  ' | yourname = 'Monty  ' |
+--------------------+----------------------+
|                  1 |                    1 |
+--------------------+----------------------+
1 row in set (0.00 sec)
```

This is true for all MySQL versions, and it makes no difference whether your version trims trailing spaces from VARCHAR values before storing them. Nor does the server SQL mode make any difference in this regard.

Note

For more information about MySQL character sets and collations, see Section 9.1, "Character Set Support".

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters will result in a duplicate-key error. For example, if a table contains `'a'`, an attempt to store `'a '` causes a duplicate-key error.

The BINARY attribute is sticky. This means that if a column marked BINARY is used in an expression, the whole expression is treated as a BINARY value.

MySQL may silently change the type of a CHAR or VARCHAR column at table creation time. See Section 12.1.5.2, "Silent Column Specification Changes".

10.4.2 The BINARY and VARBINARY Types

The BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than nonbinary strings. That is, they contain byte strings rather than character strings. This means that they have no character set, and sorting and comparison are based on the numeric values of the bytes in the values.

The permissible maximum length is the same for BINARY and VARBINARY as it is for CHAR and VARCHAR, except that the length for BINARY and VARBINARY is a length in bytes rather than in characters.

Before MySQL 4.1.2, `BINARY(M)` and `VARBINARY(M)` are treated as `CHAR(M) BINARY` and `VARCHAR(M) BINARY`. As of MySQL 4.1.2, the BINARY and VARBINARY data types are distinct from the `CHAR BINARY` and `VARCHAR BINARY` data types. For the latter types, the BINARY attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, and the column itself contains nonbinary character strings rather than binary byte strings. For example, in 4.1.2 and up, `CHAR(5) BINARY` is treated as `CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin`, assuming that the default character set is `latin1`. This differs from `BINARY(5)`, which stores 5-bytes binary strings that have no character set or collation. For information about differences between nonbinary string binary collations and binary strings, see Section 9.1.7.6, "The _bin and binary Collations".

If you assign a value to a BINARY or VARBINARY column that exceeds the column's maximum length, the value is truncated to fit. If the truncated characters are not spaces, a warning is generated.

The handling of trailing spaces is the same for BINARY and VARBINARY as it is for CHAR and VARCHAR. When BINARY values are stored, they are right-padded with spaces to the specified length. When BINARY values are retrieved, trailing spaces are removed. For VARBINARY, trailing spaces are removed when values are stored.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad bytes will result in a duplicate-key error. For example, if a table contains `'a'`, an attempt to store `'a '` causes a duplicate-key error. Trailing spaces are significant in comparisons.

You should consider the preceding padding and stripping characteristics carefully if you plan to use one of these data types for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how space-padding of `BINARY` values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET c = 'a  ';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a  ' from t;
+--------+---------+-----------+
| HEX(c) | c = 'a' | c = 'a  ' |
+--------+---------+-----------+
| 61     | 1       | 0         |
+--------+---------+-----------+
1 row in set (0.00 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use one of the `BLOB` data types instead.

10.4.3 The BLOB and TEXT Types

A `BLOB` is a binary large object that can hold a variable amount of data. The four `BLOB` types are `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, and `LONGBLOB`. These differ only in the maximum length of the values they can hold. The four `TEXT` types are `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, and `LONGTEXT`. These correspond to the four `BLOB` types and have the same maximum lengths and storage requirements. See Section 10.5, “Data Type Storage Requirements”.

`BLOB` values are treated as binary strings (byte strings). They have no character set, and sorting and comparison are based on the numeric values of the bytes in column values. `TEXT` values are treated as nonbinary strings (character strings). They have a character set, and values are sorted and compared based on the collation of the character set assigned to the column as of MySQL 4.1. Before 4.1, `TEXT` sorting and comparison are based on the collation of the server character set.

If you assign a value to a `BLOB` or `TEXT` column that exceeds the data type's maximum length, the value is truncated to fit and a warning is generated.

If a `TEXT` or `BLOB` column is indexed, index entry comparisons are not space-padded at the end. This means that, if the index requires unique values, duplicate-key errors will not occur for values that differ only in the number of trailing spaces. For example, if a table contains `'a'`, an attempt to store `'a '` does not cause a duplicate-key error. (This behavior changes in MySQL 5.0 for `TEXT` columns, such that comparisons are space-padded.)

In most respects, you can regard a `BLOB` column as a `VARBINARY` column that can be as big as you like. Similarly, you can regard a `TEXT` column as a `VARCHAR` column. `BLOB` and `TEXT` differ from `VARBINARY` and `VARCHAR` in the following ways:

- There is no trailing-space removal for `BLOB` and `TEXT` columns when values are stored or retrieved. This differs from `VARBINARY` and `VARCHAR`, for which trailing spaces are removed when values are stored.

  On comparisons, `TEXT` is space extended to fit the compared object, exactly like `CHAR` and `VARCHAR`.

- You can have indexes on BLOB and TEXT columns only as of MySQL 3.23.2 for MyISAM tables or MySQL 4.0.14 for InnoDB tables. Previous versions of MySQL did not support indexing these data types.
- For indexes on BLOB and TEXT columns, you must specify an index prefix length. For CHAR and VARCHAR, a prefix length is optional. See Section 7.4.1, “Column Indexes”.
- BLOB and TEXT columns cannot have DEFAULT values.

From MySQL 4.1.0 on, LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature. If you use the BINARY attribute with a TEXT data type, the column is assigned the binary collation of the column character set.

MySQL Connector/ODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values can be extremely long, you might encounter some constraints in using them:

- Only the first max_sort_length [420] bytes of the column are used when sorting. The default value of max_sort_length [420] is 1024. This value can be changed using the --max_sort_length=*N* [420] option when starting the mysqld server. See Section 5.1.3, “Server System Variables”.

  As of MySQL 4.0.3, you can make more bytes significant in sorting or grouping by increasing the value of max_sort_length [420] at runtime. Any client can change the value of its session max_sort_length [420] variable:

  ```
  mysql> SET max_sort_length = 2000;
  mysql> SELECT id, comment FROM t
      -> ORDER BY comment;
  ```

  Another way to use GROUP BY or ORDER BY on a BLOB or TEXT column containing long values when you want more than max_sort_length [420] bytes to be significant is to convert the column value into a fixed-length object. The standard way to do this is with the SUBSTRING() [802] function. For example, the following statement causes 2000 bytes of the comment column to be taken into account for sorting:

  ```
  mysql> SELECT id, SUBSTRING(comment,1,2000) FROM t
      -> ORDER BY SUBSTRING(comment,1,2000);
  ```

  Before MySQL 3.23.2, you can group on an expression involving BLOB or TEXT values by using a column alias or by specifying the column position:

  ```
  mysql> SELECT id, SUBSTRING(comment,1,2000) AS b
      -> FROM tbl_name GROUP BY b;
  mysql> SELECT id, SUBSTRING(comment,1,2000)
      -> FROM tbl_name GROUP BY 2;
  ```

- Instances of BLOB or TEXT columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the MEMORY storage engine does not support those data types (see Section 7.7.4, “How MySQL Uses Internal Temporary Tables”). Use of disk incurs a performance penalty, so include BLOB or TEXT columns in the query result only if they are really needed. For example, avoid using SELECT *, which selects all columns.
- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing

the value of the max_allowed_packet [418] variable, but you must do so for both the server and your client program. For example, both mysql and mysqldump enable you to change the client-side max_allowed_packet [418] value. See Section 7.8.2, “Tuning Server Parameters”, Section 4.5.1, “mysql — The MySQL Command-Line Tool”, and Section 4.5.4, “mysqldump — A Database Backup Program”. You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see Section 10.5, “Data Type Storage Requirements”

Each BLOB or TEXT value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in BLOB or TEXT columns. You may find MySQL's string handling functions useful for working with such data. See Section 11.5, “String Functions”. For security and other reasons, it is usually preferable to do so using application code rather than giving application users the FILE [491] privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (http://forums.mysql.com/).

10.4.4 The ENUM Type

An ENUM is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time.

An enumeration value must be a quoted string literal; it may not be an expression, even one that evaluates to a string value. For example, you can create a table with an ENUM column like this:

```
CREATE TABLE sizes (
    name ENUM('small', 'medium', 'large')
);
```

However, this version of the previous CREATE TABLE statement does *not* work:

```
CREATE TABLE sizes (
    c1 ENUM('small', CONCAT('med','ium'), 'large')
);
```

You also may not employ a user variable as an enumeration value. This pair of statements do *not* work:

```
SET @mysize = 'medium';

CREATE TABLE sizes (
    name ENUM('small', @mysize, 'large')
);
```

If you wish to use a number as an enumeration value, you must enclose it in quotation marks.

The value may also be the empty string (`''`) or NULL under certain circumstances:

- If you insert an invalid value into an ENUM (that is, a string not present in the list of permitted values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numeric value 0. More about this later.

  If strict SQL mode is enabled, attempts to insert invalid ENUM values result in an error.

- If an ENUM column is declared to permit NULL, the NULL value is a legal value for the column, and the default value is NULL. If an ENUM column is declared NOT NULL, its default value is the first element of the list of permitted values.

Each enumeration value has an index:

- Values from the list of permissible elements in the column specification are numbered beginning with 1.
- The index value of the empty string error value is 0. This means that you can use the following SELECT statement to find rows into which invalid ENUM values were assigned:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the NULL value is NULL.
- The term “index” here refers only to position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as ENUM('one', 'two', 'three') can have any of the values shown here. The index of each value is also shown.

Value	Index
NULL	NULL
''	0
'one'	1
'two'	2
'three'	3

An enumeration can have a maximum of 65,535 elements.

Starting from MySQL 3.23.51, trailing spaces are automatically deleted from ENUM member values in the table definition when a table is created.

When retrieved, values stored into an ENUM column are displayed using the lettercase that was used in the column definition. Before MySQL 4.1.1, lettercase is irrelevant when you assign values to an ENUM column. As of 4.1.1, ENUM columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase does matter when you assign values to the column.

If you retrieve an ENUM value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an ENUM column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

If you store a number into an ENUM column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does *not* work with LOAD DATA, which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an ENUM column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of '0', '1', and '2', but numeric index values of 1, 2, and 3:

```
numbers ENUM('0','1','2')
```

If you store 2, it is interpreted as an index value, and becomes '1' (the value with index 2). If you store '2', it matches an enumeration value, so it is stored as '2'. If you store '3', it does not match any enumeration value, so it is treated as an index and becomes '2' (the value with index 3).

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
```

```
mysql> SELECT * FROM t;
+---------+
| numbers |
+---------+
| 1       |
| 2       |
| 2       |
+---------+
```

ENUM values are sorted according to the order in which the enumeration members were listed in the column specification. (In other words, ENUM values are sorted according to their index numbers.) For example, 'a' sorts before 'b' for ENUM('a', 'b'), but 'b' sorts before 'a' for ENUM('b', 'a'). The empty string sorts before nonempty strings, and NULL values sort before all other enumeration values. If you expect sorting to be done alphabetically, you should specify the ENUM list in alphabetic order. You can also use GROUP BY CAST(col AS CHAR) or GROUP BY CONCAT(col) to make sure that the column is sorted lexically rather than by index number.

Functions such as SUM() [884] or AVG() [881] that expect a numeric argument cast the argument to a number if necessary. For ENUM values, the cast operation causes the index number to be used.

If you want to determine all possible values for an ENUM column, use SHOW COLUMNS FROM *tbl_name* LIKE *enum_col* and parse the ENUM definition in the Type column of the output.

10.4.5 The SET Type

A SET is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (“,”). A consequence of this is that SET member values should not themselves contain commas.

For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:

```
''
'one'
'two'
'one,two'
```

A SET can have a maximum of 64 different members.

Starting from MySQL 3.23.51, trailing spaces are automatically deleted from SET member values in the table definition when a table is created.

When retrieved, values stored into a SET column are displayed using the lettercase that was used in the column definition. Before MySQL 4.1.1, lettercase is irrelevant when you assign values to an SET column. As of 4.1.1, SET columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase does matter when you assign values to the column.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a SET value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a SET column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a','b','c','d'), the members have the following decimal and binary values.

SET Member	Decimal Value	Binary Value
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, the first and fourth SET members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. For example, suppose that a column is specified as SET('a','b','c','d'):

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Then all these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
5 rows in set (0.04 sec)
```

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+---------+------+-------------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
```

```
| a,d  |
| a,d  |
| a,d  |
+------+
6 rows in set (0.01 sec)
```

If strict SQL mode is enabled, attempts to insert invalid `SET` values result in an error.

`SET` values are sorted numerically. `NULL` values sort before non-`NULL` `SET` values.

Functions such as `SUM()` [884] or `AVG()` [881] that expect a numeric argument cast the argument to a number if necessary. For `SET` values, the cast operation causes the numeric value to be used.

Normally, you search for `SET` values using the `FIND_IN_SET()` [796] function or the `LIKE` [804] operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where *`set_col`* contains the *`value`* set member. The second is similar, but not the same: It finds rows where *`set_col`* contains *`value`* anywhere, even as a substring of another set member.

The following statements also are legal:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to *`'val1,val2'`* returns different results than comparing values to *`'val2,val1'`*. You should specify the values in the same order in which they are listed in the column definition.

If you want to determine all possible values for a `SET` column, use `SHOW COLUMNS FROM` *`tbl_name`* `LIKE` *`set_col`* and parse the `SET` definition in the `Type` column of the output.

10.5 Data Type Storage Requirements

The storage requirements for each data type supported by MySQL are listed here by category.

The maximum size of a row in a `MyISAM` table is 65,535 bytes. (However, each `BLOB` or `TEXT` column contributes only 9 to 12 bytes toward this size.) This limitation may be shared by other storage engines as well. See Chapter 13, *Storage Engines*, for more information.

Important

For tables using the `NDBCLUSTER` storage engine, there is the factor of *4-byte alignment* to be taken into account when calculating storage requirements. This means that all `NDB` data storage is done in multiples of 4 bytes. Thus, a column value that would take 15 bytes in a table using a storage engine other than `NDB` requires 16 bytes in an `NDB` table. This requirement applies in addition to any other considerations that are discussed in this section. For example, in `NDBCLUSTER` tables, the `TINYINT`, `SMALLINT`, `MEDIUMINT`, and `INTEGER` (`INT`) column types each require 4 bytes storage per record due to the alignment factor.

An exception to this rule is the `BIT` type, which is *not* 4-byte aligned. In MySQL Cluster tables, a `BIT(`*`M`*`)` column takes *`M`* bits of storage space. However, if a

table definition contains 1 or more `BIT` columns (up to 32 `BIT` columns), then `NDBCLUSTER` reserves 4 bytes (32 bits) per row for these. If a table definition contains more than 32 `BIT` columns (up to 64 such columns), then `NDBCLUSTER` reserves 8 bytes (that is, 64 bits) per row.

In addition, while a `NULL` itself does not require any storage space, `NDBCLUSTER` reserves 4 bytes per row if the table definition contains any columns defined as `NULL`, up to 32 `NULL` columns. (If a MySQL Cluster table is defined with more than 32 `NULL` columns up to 64 `NULL` columns, then 8 bytes per row is reserved.)

When calculating storage requirements for MySQL Cluster tables, you must also remember that every table using the `NDBCLUSTER` storage engine requires a primary key; if no primary key is defined by the user, then a “hidden” primary key will be created by `NDB`. This hidden primary key consumes 31-35 bytes per table record.

You may find the `ndb_size.pl` utility to be useful for estimating `NDB` storage requirements. This Perl script connects to a current MySQL (non-Cluster) database and creates a report on how much space that database would require if it used the `NDBCLUSTER` storage engine. See Section 15.4.18, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”, for more information.

Storage Requirements for Numeric Types

Data Type	Storage Required
`TINYINT`	1 byte
`SMALLINT`	2 bytes
`MEDIUMINT`	3 bytes
`INT`, `INTEGER`	4 bytes
`BIGINT`	8 bytes
`FLOAT(`*p*`)`	4 bytes if 0 <= *p* <= 24, 8 bytes if 25 <= *p* <= 53
`FLOAT`	4 bytes
`DOUBLE [PRECISION]`, `REAL`	8 bytes
`DECIMAL(`*M*`,`*D*`)`, `NUMERIC(`*M*`,`*D*`)`	Varies; see following discussion

In MySQL versions up to and including 4.1, `DECIMAL` columns are represented as strings and their storage requirements are:

- *M*+2 bytes, if *D* > 0
- *M*+1 bytes, if *D* = 0
- *D*+2, if *M* < *D*

Storage Requirements for Date and Time Types

Data Type	Storage Required
`DATE`	3 bytes
`TIME`	3 bytes
`DATETIME`	8 bytes
`TIMESTAMP`	4 bytes

Data Type	Storage Required
YEAR	1 byte

For details about internal representation of temporal values, see MySQL Internals: Important Algorithms and Structures.

Storage Requirements for String Types

In the following table, *M* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *L* represents the actual length in bytes of a given string value.

Data Type	Storage Required
CHAR(*M*)	*M* × *w* bytes, 0 <= *M* <= 255, where *w* is the number of bytes required for the maximum-length character in the character set
BINARY(*M*)	*M* bytes, 0 <= *M* <= 255
VARCHAR(*M*), VARBINARY(*M*)	*L* + 1 bytes, 0 <= *L* <= 255
TINYBLOB, TINYTEXT	*L* + 1 bytes, where $L < 2^8$
BLOB, TEXT	*L* + 2 bytes, where $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	*L* + 3 bytes, where $L < 2^{24}$
LONGBLOB, LONGTEXT	*L* + 4 bytes, where $L < 2^{32}$
ENUM('*value1*','*value2*',...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('*value1*','*value2*',...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is *L* (the byte length of the string). For example, storage for a MEDIUMTEXT value requires *L* bytes to store the value plus three bytes to store the length of the value.

As of MySQL 4.1, to calculate the number of bytes used to store a particular CHAR, VARCHAR, or TEXT column value, you must take into account the character set used for that column and whether the value contains multi-byte characters. In particular, when using the utf8 Unicode character set, you must keep in mind that not all characters use the same number of bytes and can require up to three bytes per character. For a breakdown of the storage used for different categories of utf8 characters, see Section 9.1.9, "Unicode Support".

VARCHAR and the BLOB and TEXT types are variable-length types. For each, the storage requirements depend on the actual length of column values (represented by *L* in the preceding table), rather than on the type's maximum possible size. For example, a VARCHAR(10) column can hold a string with a maximum length of 10 characters. The actual storage required is the length of the string (*L*), plus one byte to record the length of the string. For the string 'abcd', *L* is 4 and the storage requirement is five bytes.

Note

The NDBCLUSTER engine supports only fixed-width columns. This means that a VARCHAR column from a table in a MySQL Cluster will behave almost as if it were of type CHAR (except that each record still has one extra byte overhead). For example, in an NDB table, *each* record in a column declared as VARCHAR(100) will take up 101 bytes for storage, regardless of the length of the string actually stored in any given record.

`TEXT` and `BLOB` columns are implemented differently in the `NDBCLUSTER` storage engine, wherein each record in a `TEXT` column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The records in this second table are always 2,000 bytes long. This means that the size of a `TEXT` column is 256 if *`size`* <= 256 (where *`size`* represents the size of the record); otherwise, the size is `256 + size + (2000 - (size - 256) % 2000)`.

The size of an `ENUM` object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See Section 10.4.4, “The ENUM Type”.

The size of a `SET` object is determined by the number of different set members. If the set size is *`N`*, the object occupies `(N + 7)/8` bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A `SET` can have a maximum of 64 members. See Section 10.4.5, “The SET Type”.

10.6 Choosing the Right Type for a Column

For the most efficient use of storage, try to use the most precise type in all cases. For example, if an integer column is used for values in the range from `1` to `99999`, `MEDIUMINT UNSIGNED` is the best type. Of the types that represent all the required values, it uses the least amount of storage.

For earlier MySQL versions, accurate representation of monetary values was a common problem. In these MySQL versions, you should also use the `DECIMAL` type. In this case the value is stored as a string, so no loss of accuracy should occur on storage. However, calculations on these `DECIMAL` values are done using double-precision operations. If accuracy is not too important or if speed is important, the `DOUBLE` type may also be good enough.

For high precision, you can always convert to a fixed-point type stored in a `BIGINT`. This enables you to do all calculations with 64-bit integers and then convert results back to floating-point values only when necessary.

`PROCEDURE ANALYSE` can be used to obtain suggestions for optimal column data types. For more information, see Section 18.3.1, “PROCEDURE ANALYSE”.

10.7 Using Data Types from Other Database Engines

To make it easier to use code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL.

Other Vendor Type	MySQL Type
`BINARY(M)`	`CHAR(M) BINARY` (before MySQL 4.1.2)
`BOOL`	`TINYINT`
`BOOLEAN`	`TINYINT`
`CHARACTER VARYING(M)`	`VARCHAR(M)`
`FIXED`	`DECIMAL` (MySQL 4.1.0 on)
`FLOAT4`	`FLOAT`
`FLOAT8`	`DOUBLE`
`INT1`	`TINYINT`
`INT2`	`SMALLINT`

Other Vendor Type	MySQL Type
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT (MySQL 4.1.0 on)
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL
VARBINARY(*M*)	VARCHAR(*M*) BINARY (before MySQL 4.1.2)

As of MySQL 4.1.2, BINARY and VARBINARY are distinct data types and are not converted to CHAR BINARY and VARCHAR BINARY.

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a DESCRIBE *tbl_name* statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE t;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| a     | tinyint(1)    | YES  |     | NULL    |       |
| b     | double        | YES  |     | NULL    |       |
| c     | mediumtext    | YES  |     | NULL    |       |
| d     | decimal(10,0) | YES  |     | NULL    |       |
+-------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Chapter 11 Functions and Operators

Table of Contents

Expressions can be used at several points in SQL statements, such as in the `ORDER BY` or `HAVING` clauses of `SELECT` statements, in the `WHERE` clause of a `SELECT`, `DELETE`, or `UPDATE` statement, or in `SET` statements. Expressions can be written using literal values, column values, `NULL`, built-in functions, user-defined functions, and operators. This chapter describes the functions and operators that are permitted for writing expressions in MySQL. Instructions for writing user-defined functions are given in Section 18.2, “Adding New Functions to MySQL”. See Section 8.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for a particular function or operator.

Note

By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function

calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` [394] option. (See Section 5.1.6, “Server SQL Modes”.) Individual client programs can request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the `mysql` program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
        -> 2
```

11.1 Function and Operator Reference

Note

This table is part of an ongoing process to expand and simplify the information provided on these elements. Further improvements to the table, and corresponding descriptions will be applied over the coming months.

Table 11.1 Functions/Operators

Name	Description
`ABS()` [817]	Return the absolute value
`ACOS()` [817]	Return the arc cosine
`ADDDATE()` [827]	Add time values (intervals) to a date value
`ADDTIME()` [828]	Add time
`AES_DECRYPT()` [865]	Decrypt using AES
`AES_ENCRYPT()` [865]	Encrypt using AES
`AND`, `&&` [787]	Logical AND
`ASCII()` [793]	Return numeric value of left-most character
`ASIN()` [818]	Return the arc sine
`=` [789]	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
`:=` [788]	Assign a value
`ATAN2()`, `ATAN()` [818]	Return the arc tangent of the two arguments
`ATAN()` [818]	Return the arc tangent
`AVG()` [881]	Return the average value of the argument
`BENCHMARK()` [870]	Repeatedly execute an expression

Name	Description
BETWEEN ... AND ... [783]	Check whether a value is within a range of values
BIN() [794]	Return a string containing binary representation of a number
BINARY [859]	Cast a string to a binary string
BIT_AND() [882]	Return bitwise and
BIT_COUNT() [863]	Return the number of bits that are set
BIT_LENGTH() [794]	Return length of argument in bits
BIT_OR() [882]	Return bitwise or
BIT_XOR() [882]	Return bitwise xor
& [863]	Bitwise AND
~ [863]	Invert bits
| [862]	Bitwise OR
^ [863]	Bitwise XOR
CASE [790]	Case operator
CAST() [859]	Cast a value as a certain type
CEIL() [818]	Return the smallest integer value not less than the argument
CEILING() [818]	Return the smallest integer value not less than the argument
CHAR_LENGTH() [794]	Return number of characters in argument
CHAR() [794]	Return the character for each integer passed
CHARACTER_LENGTH() [794]	Synonym for CHAR_LENGTH()
CHARSET() [870]	Return the character set of the argument
COALESCE() [784]	Return the first non-NULL argument
COERCIBILITY() [871]	Return the collation coercibility value of the string argument
COLLATION() [871]	Return the collation of the string argument
COMPRESS() [866]	Return result as a binary string
CONCAT_WS() [795]	Return concatenate with separator
CONCAT() [795]	Return concatenated string
CONNECTION_ID() [872]	Return the connection ID (thread ID) for the connection
CONV() [818]	Convert numbers between different number bases
CONVERT_TZ() [828]	Convert from one timezone to another
CONVERT() [859]	Cast a value as a certain type
COS() [819]	Return the cosine
COT() [819]	Return the cotangent
COUNT(DISTINCT) [882]	Return the count of a number of different values
COUNT() [882]	Return a count of the number of rows returned
CRC32() [819]	Compute a cyclic redundancy check value
CURDATE() [828]	Return the current date
CURRENT_DATE(), CURRENT_DATE [829]	Synonyms for CURDATE()

Name	Description
CURRENT_TIME(), CURRENT_TIME [829]	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP [829]	Synonyms for NOW()
CURRENT_USER(), CURRENT_USER [872]	The authenticated user name and host name
CURTIME() [829]	Return the current time
DATABASE() [872]	Return the default (current) database name
DATE_ADD() [829]	Add time values (intervals) to a date value
DATE_FORMAT() [832]	Format date as specified
DATE_SUB() [833]	Subtract a time value (interval) from a date
DATE() [829]	Extract the date part of a date or datetime expression
DATEDIFF() [829]	Subtract two dates
DAY() [833]	Synonym for DAYOFMONTH()
DAYNAME() [833]	Return the name of the weekday
DAYOFMONTH() [833]	Return the day of the month (0-31)
DAYOFWEEK() [834]	Return the weekday index of the argument
DAYOFYEAR() [834]	Return the day of the year (1-366)
DECODE() [866]	Decodes a string encrypted using ENCODE()
DEFAULT() [877]	Return the default value for a table column
DEGREES() [819]	Convert radians to degrees
DES_DECRYPT() [866]	Decrypt a string
DES_ENCRYPT() [866]	Encrypt a string
DIV [816]	Integer division
/ [816]	Division operator
ELT() [795]	Return string at index number
ENCODE() [867]	Encode a string
ENCRYPT() [867]	Encrypt a string
<=> [782]	NULL-safe equal to operator
= [782]	Equal operator
EXP() [819]	Raise to the power of
EXPORT_SET() [795]	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
EXTRACT() [834]	Extract part of a date
FIELD() [796]	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET() [796]	Return the index position of the first argument within the second argument
FLOOR() [820]	Return the largest integer value not greater than the argument

Name	Description
FORMAT() [796]	Return a number formatted to specified number of decimal places
FOUND_ROWS() [872]	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
FROM_DAYS() [834]	Convert a day number to a date
FROM_UNIXTIME() [834]	Format UNIX timestamp as a date
GET_FORMAT() [835]	Return a date format string
GET_LOCK() [877]	Get a named lock
>= [782]	Greater than or equal operator
> [783]	Greater than operator
GREATEST() [784]	Return the largest argument
GROUP_CONCAT() [883]	Return a concatenated string
HEX() [796]	Return a hexadecimal representation of a decimal or string value
HOUR() [836]	Extract the hour
IF() [790]	If/else construct
IFNULL() [791]	Null if/else construct
IN() [784]	Check whether a value is within a set of values
INET_ATON() [878]	Return the numeric value of an IP address
INET_NTOA() [878]	Return the IP address from a numeric value
INSERT() [797]	Insert a substring at the specified position up to the specified number of characters
INSTR() [797]	Return the index of the first occurrence of substring
INTERVAL() [785]	Return the index of the argument that is less than the first argument
IS_FREE_LOCK() [879]	Checks whether the named lock is free
IS NOT NULL [783]	NOT NULL value test
IS NULL [783]	NULL value test
IS_USED_LOCK() [879]	Checks whether the named lock is in use. Return connection identifier if true.
ISNULL() [785]	Test whether the argument is NULL
LAST_DAY [836]	Return the last day of the month for the argument
LAST_INSERT_ID() [874]	Value of the AUTOINCREMENT column for the last INSERT
LCASE() [797]	Synonym for LOWER()
LEAST() [786]	Return the smallest argument
<< [863]	Left shift
LEFT() [797]	Return the leftmost number of characters as specified
LENGTH() [797]	Return the length of a string in bytes
<= [782]	Less than or equal operator

Name	Description
< [782]	Less than operator
LIKE [804]	Simple pattern matching
LN() [820]	Return the natural logarithm of the argument
LOAD_FILE() [798]	Load the named file
LOCALTIME(), LOCALTIME [836]	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP() [836]	Synonym for NOW()
LOCATE() [798]	Return the position of the first occurrence of substring
LOG10() [821]	Return the base-10 logarithm of the argument
LOG2() [821]	Return the base-2 logarithm of the argument
LOG() [820]	Return the natural logarithm of the first argument
LOWER() [798]	Return the argument in lowercase
LPAD() [798]	Return the string argument, left-padded with the specified string
LTRIM() [799]	Remove leading spaces
MAKE_SET() [799]	Return a set of comma-separated strings that have the corresponding bit in bits set
MAKEDATE() [836]	Create a date from the year and day of year
MAKETIME() [836]	Create time from hour, minute, second
MASTER_POS_WAIT() [879]	Block until the slave has read and applied all updates up to the specified position
MATCH [845]	Perform full-text search
MAX() [884]	Return the maximum value
MD5() [868]	Calculate MD5 checksum
MICROSECOND() [837]	Return the microseconds from argument
MID() [799]	Return a substring starting from the specified position
MIN() [884]	Return the minimum value
- [815]	Minus operator
MINUTE() [837]	Return the minute from the argument
MOD() [821]	Return the remainder
% or MOD [816]	Modulo operator
MONTH() [837]	Return the month from the date passed
MONTHNAME() [837]	Return the name of the month
NOT BETWEEN ... AND ... [784]	Check whether a value is not within a range of values
!=, <> [782]	Not equal operator
NOT IN() [785]	Check whether a value is not within a set of values
NOT LIKE [806]	Negation of simple pattern matching
NOT REGEXP [808]	Negation of REGEXP
NOT, ! [787]	Negates value

Name	Description
NOW() [837]	Return the current date and time
NULLIF() [791]	Return NULL if expr1 = expr2
OCT() [821]	Return a string containing octal representation of a number
OCTET_LENGTH() [799]	Synonym for LENGTH()
OLD_PASSWORD() [868]	Return the value of the pre-4.1 implementation of PASSWORD
||, OR [787]	Logical OR
ORD() [799]	Return character code for leftmost character of the argument
PASSWORD() [868]	Calculate and return a password string
PERIOD_ADD() [837]	Add a period to a year-month
PERIOD_DIFF() [838]	Return the number of months between periods
PI() [822]	Return the value of pi
+ [815]	Addition operator
POSITION() [799]	Synonym for LOCATE()
POW() [822]	Return the argument raised to the specified power
POWER() [822]	Return the argument raised to the specified power
PROCEDURE ANALYSE()	Analyze the results of a query
QUARTER() [838]	Return the quarter from a date argument
QUOTE() [800]	Escape the argument for use in an SQL statement
RADIANS() [822]	Return argument converted to radians
RAND() [822]	Return a random floating-point value
REGEXP [808]	Pattern matching using regular expressions
RELEASE_LOCK() [879]	Releases the named lock
REPEAT() [800]	Repeat a string the specified number of times
REPLACE() [800]	Replace occurrences of a specified string
REVERSE() [800]	Reverse the characters in a string
>> [863]	Right shift
RIGHT() [800]	Return the specified rightmost number of characters
RLIKE [808]	Synonym for REGEXP
ROUND() [823]	Round the argument
RPAD() [800]	Append string the specified number of times
RTRIM() [801]	Remove trailing spaces
SEC_TO_TIME() [838]	Converts seconds to 'HH:MM:SS' format
SECOND() [838]	Return the second (0-59)
SESSION_USER() [876]	Synonym for USER()
SHA1(), SHA() [869]	Calculate an SHA-1 160-bit checksum
SIGN() [824]	Return the sign of the argument
SIN() [824]	Return the sine of the argument
SOUNDEX() [801]	Return a soundex string

Name	Description
SOUNDS LIKE [801]	Compare sounds
SPACE() [801]	Return a string of the specified number of spaces
SQRT() [824]	Return the square root of the argument
STD() [884]	Return the population standard deviation
STDDEV() [884]	Return the population standard deviation
STR_TO_DATE() [838]	Convert a string to a date
STRCMP() [807]	Compare two strings
SUBDATE() [839]	Synonym for DATE_SUB() when invoked with three arguments
SUBSTR() [802]	Return the substring as specified
SUBSTRING_INDEX() [802]	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING() [802]	Return the substring as specified
SUBTIME() [840]	Subtract times
SUM() [884]	Return the sum
SYSDATE() [840]	Return the time at which the function executes
SYSTEM_USER() [876]	Synonym for USER()
TAN() [824]	Return the tangent of the argument
TIME_FORMAT() [840]	Format as time
TIME_TO_SEC() [841]	Return the argument converted to seconds
TIME() [840]	Extract the time portion of the expression passed
TIMEDIFF() [840]	Subtract time
* [816]	Multiplication operator
TIMESTAMP() [840]	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TO_DAYS() [841]	Return the date argument converted to days
TRIM() [803]	Remove leading and trailing spaces
TRUNCATE() [825]	Truncate to specified number of decimal places
UCASE() [803]	Synonym for UPPER()
- [815]	Change the sign of the argument
UNCOMPRESS() [869]	Uncompress a string compressed
UNCOMPRESSED_LENGTH() [869]	Return the length of a string before compression
UNHEX() [803]	Return a string containing hex representation of a number
UNIX_TIMESTAMP() [841]	Return a UNIX timestamp
UPPER() [804]	Convert to uppercase
USER() [876]	The user name and host name provided by the client
UTC_DATE() [842]	Return the current UTC date
UTC_TIME() [842]	Return the current UTC time
UTC_TIMESTAMP() [843]	Return the current UTC date and time

Name	Description
UUID() [879]	Return a Universal Unique Identifier (UUID)
VALUES() [880]	Defines the values to be used during an INSERT
VARIANCE() [884]	Return the population standard variance
VERSION() [876]	Returns a string that indicates the MySQL server version
WEEK() [843]	Return the week number
WEEKDAY() [844]	Return the weekday index
WEEKOFYEAR() [844]	Return the calendar week of the date (0-53)
XOR [788]	Logical XOR
YEAR() [844]	Return the year
YEARWEEK() [844]	Return the year and week

11.2 Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts numbers to strings as necessary, and vice versa.

```
mysql> SELECT 1+'1';
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

It is also possible to convert a number to a string explicitly using the CAST() [859] function. Conversion occurs implicitly with the CONCAT() [795] function because it expects string arguments. (CAST() [859] is preferable, but is unavailable before MySQL 4.0.2.)

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
        -> 38.8, '38.8'
```

See later in this section for information about the character set of implicit number-to-string conversions.

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are NULL, the result of the comparison is NULL, except for the NULL-safe <=> [782] equality comparison operator. For NULL <=> NULL, the result is true. No conversion is needed.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a TIMESTAMP or DATETIME column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. Note that this is not done for the arguments to IN() [784]! To be safe, always use complete datetime, date, or time strings when doing comparisons. For example, to achieve best results when using BETWEEN [783] with date or time values, use CAST() [859] to explicitly convert the values to the desired data type.

- In all other cases, the arguments are compared as floating-point (real) numbers.

For information about conversion of values from one temporal type to another, see Section 10.3.5, “Conversion Between Date and Time Types”.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

For comparisons of a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If *`str_col`* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value `1`, such as `'1'`, `' 1'`, or `'1a'`.

Comparisons that use floating-point numbers (or values that are converted to floating-point numbers) are approximate because such numbers are inexact. This might lead to results that appear inconsistent:

```
mysql> SELECT '18015376320243458' = 18015376320243458;
        -> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
        -> 0
```

Such results can occur because the values are converted to floating-point numbers, which have only 53 bits of precision and are subject to rounding:

```
mysql> SELECT '18015376320243459'+0.0;
        -> 1.8015376320243e+16
```

Furthermore, the conversion from string to floating-point and from integer to floating-point do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications.

The results shown will vary on different systems, and can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use `CAST()` [859] so that a value will not be converted implicitly to a float-point number:

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243459;
        -> 1
```

For more information about floating-point comparisons, see Section B.5.5.8, “Problems with Floating-Point Values”.

Implicit conversion of a numeric or temporal value to a string produces a binary string (a `BINARY`, `VARBINARY`, or `BLOB` value). Such implicit conversions to string typically occur for functions that are passed numeric or temporal values when string values are more usual, and thus can have effects beyond

the type of the converted value. Consider the expression `CONCAT(1, 'abc')` [795]. The numeric argument `1` is converted to the binary string `'1'` and the concatenation of that value with the nonbinary string `'abc'` produces the binary string `'1abc'`.

11.3 Operators

Table 11.2 Operators

Name	Description
`AND`, `&&` [787]	Logical AND
`=` [789]	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
`:=` [788]	Assign a value
`BETWEEN ... AND ...` [783]	Check whether a value is within a range of values
`BINARY` [859]	Cast a string to a binary string
`&` [863]	Bitwise AND
`~` [863]	Invert bits
`|` [862]	Bitwise OR
`^` [863]	Bitwise XOR
`CASE` [790]	Case operator
`DIV` [816]	Integer division
`/` [816]	Division operator
`<=>` [782]	NULL-safe equal to operator
`=` [782]	Equal operator
`>=` [782]	Greater than or equal operator
`>` [783]	Greater than operator
`IS NOT NULL` [783]	NOT NULL value test
`IS NULL` [783]	NULL value test
`<<` [863]	Left shift
`<=` [782]	Less than or equal operator
`<` [782]	Less than operator
`LIKE` [804]	Simple pattern matching
`-` [815]	Minus operator
`% or MOD` [816]	Modulo operator
`NOT BETWEEN ... AND ...` [784]	Check whether a value is not within a range of values
`!=`, `<>` [782]	Not equal operator
`NOT LIKE` [806]	Negation of simple pattern matching
`NOT REGEXP` [808]	Negation of REGEXP
`NOT`, `!` [787]	Negates value
`||`, `OR` [787]	Logical OR
`+` [815]	Addition operator
`REGEXP` [808]	Pattern matching using regular expressions

Name	Description
>> [863]	Right shift
RLIKE [808]	Synonym for REGEXP
SOUNDS LIKE [801]	Compare sounds
* [816]	Multiplication operator
- [815]	Change the sign of the argument
XOR [788]	Logical XOR

11.3.1 Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
INTERVAL
BINARY, COLLATE
!, NOT
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
&&, AND
XOR
||, OR
= (assignment), :=
```

The precedence of = depends on whether it is used as a comparison operator (= [782]) or as an assignment operator (= [789]). When used as a comparison operator, it has the same precedence as <=> [782], >= [782], > [783], <= [782], < [782], <> [782], != [782], IS, LIKE [804], REGEXP [808], and IN [784]. When used as an assignment operator, it has the same precedence as := [788]. Section 12.4.4, “SET Syntax”, and Section 8.4, “User-Defined Variables”, explain how MySQL determines which interpretation of = should apply.

The || [787] operator has a precedence between ^ [863] and the unary operators if the PIPES_AS_CONCAT [460] SQL mode is enabled.

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```
mysql> SELECT 1+2*3;
        -> 7
mysql> SELECT (1+2)*3;
        -> 9
```

11.3.2 Comparison Functions and Operators

Table 11.3 Comparison Operators

Name	Description
BETWEEN ... AND ... [783]	Check whether a value is within a range of values

Name	Description
COALESCE() [784]	Return the first non-NULL argument
<=> [782]	NULL-safe equal to operator
= [782]	Equal operator
>= [782]	Greater than or equal operator
> [783]	Greater than operator
GREATEST() [784]	Return the largest argument
IN() [784]	Check whether a value is within a set of values
INTERVAL() [785]	Return the index of the argument that is less than the first argument
IS NOT NULL [783]	NOT NULL value test
IS NULL [783]	NULL value test
ISNULL() [785]	Test whether the argument is NULL
LEAST() [786]	Return the smallest argument
<= [782]	Less than or equal operator
< [782]	Less than operator
LIKE [804]	Simple pattern matching
NOT BETWEEN ... AND ... [784]	Check whether a value is not within a range of values
!=, <> [782]	Not equal operator
NOT IN() [785]	Check whether a value is not within a set of values
NOT LIKE [806]	Negation of simple pattern matching
STRCMP() [807]	Compare two strings

Comparison operations result in a value of 1 (TRUE), 0 (FALSE), or NULL. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

The following relational comparison operators can be used to compare not only scalar operands, but row operands:

```
= > < >= <= <> !=
```

For examples of row comparisons, see Section 12.2.8.5, “Row Subqueries”.

Some of the functions in this section (such as LEAST() [786] and GREATEST() [784]) return values other than 1 (TRUE), 0 (FALSE), or NULL. However, the value they return is based on comparison operations performed according to the rules described in Section 11.2, “Type Conversion in Expression Evaluation”.

To convert a value to a specific type for comparison purposes, you can use the CAST() [859] function. String values can be converted to a different character set using CONVERT() [859]. See Section 11.10, “Cast Functions and Operators”.

By default, string comparisons are not case sensitive and use the current character set. The default is latin1 (cp1252 West European), which also works well for English.

- = [782]

  Equal:

```
mysql> SELECT 1 = 0;
        -> 0
mysql> SELECT '0' = 0;
        -> 1
mysql> SELECT '0.0' = 0;
        -> 1
mysql> SELECT '0.01' = 0;
        -> 0
mysql> SELECT '.01' = 0.01;
        -> 1
```

- <=> [782]

  `NULL`-safe equal. This operator performs an equality comparison like the = [782] operator, but returns `1` rather than `NULL` if both operands are `NULL`, and `0` rather than `NULL` if one operand is `NULL`.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
        -> 1, NULL, NULL
```

  <=> was added in MySQL 3.23.0.

- <> [782], != [782]

  Not equal:

```
mysql> SELECT '.01' <> '0.01';
        -> 1
mysql> SELECT .01 <> '0.01';
        -> 0
mysql> SELECT 'zapp' <> 'zappp';
        -> 1
```

- <= [782]

  Less than or equal:

```
mysql> SELECT 0.1 <= 2;
        -> 1
```

- < [782]

  Less than:

```
mysql> SELECT 2 < 2;
        -> 0
```

- >= [782]

  Greater than or equal:

```
mysql> SELECT 2 >= 2;
        -> 1
```

- `>` [783]

  Greater than:

```
mysql> SELECT 2 > 2;
        -> 0
```

- `IS NULL` [783]

  Tests whether a value is `NULL`.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
        -> 0, 0, 1
```

  To work well with ODBC programs, MySQL supports the following extra features when using `IS NULL` [783]:

  - If `sql_auto_is_null` [428] variable is set to 1 (the default), then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

    If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` [874] function. For details, including the return value after a multiple-row insert, see Section 11.13, “Information Functions”. If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

    The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` [783] comparison can be disabled by setting `sql_auto_is_null = 0` [428]. See Section 5.1.3, “Server System Variables”.

  - For `DATE` and `DATETIME` columns that are declared as `NOT NULL`, you can find the special date `'0000-00-00'` by using a statement like this:

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

    This is needed to get some ODBC applications to work because ODBC does not support a `'0000-00-00'` date value.

    See Obtaining Auto-Increment Values, and the description for the `FLAG_AUTO_IS_NULL` option at Connector/ODBC Connection Parameters.

- `IS NOT NULL` [783]

  Tests whether a value is not `NULL`.

```
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
        -> 1, 1, 0
```

- *`expr`* `BETWEEN` *`min`* `AND` *`max`* [783]

  If *`expr`* is greater than or equal to *`min`* and *`expr`* is less than or equal to *`max`*, `BETWEEN` [783] returns `1`, otherwise it returns `0`. This is equivalent to the expression `(`*`min`* `<=` *`expr`* `AND` *`expr`* `<=` *`max`*`)` if all the arguments are of the same type. Otherwise type conversion takes place according to the rules

described in Section 11.2, “Type Conversion in Expression Evaluation”, but applied to all the three arguments.

Note

Before MySQL 4.0.5, arguments were converted to the type of *expr* instead.

```
mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 and 1;
        -> 1, 0
mysql> SELECT 1 BETWEEN 2 AND 3;
        -> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
        -> 0
```

For best results when using BETWEEN [783] with date or time values, use CAST() [859] to explicitly convert the values to the desired data type. Examples: If you compare a DATETIME to two DATE values, convert the DATE values to DATETIME values. If you use a string constant such as '2001-1-1' in a comparison to a DATE, cast the string to a DATE.

- *expr* NOT BETWEEN *min* AND *max* [784]

  This is the same as NOT (*expr* BETWEEN *min* AND *max*).

- COALESCE(*value*,...) [784]

  Returns the first non-NULL value in the list, or NULL if there are no non-NULL values.

```
mysql> SELECT COALESCE(NULL,1);
        -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
        -> NULL
```

  COALESCE() [784] was added in MySQL 3.23.3.

- GREATEST(*value1*,*value2*,...) [784]

  With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for LEAST() [786].

```
mysql> SELECT GREATEST(2,0);
        -> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> SELECT GREATEST('B','A','C');
        -> 'C'
```

  GREATEST() [784] returns NULL only if all arguments are NULL.

  Before MySQL 3.22.5, you can use MAX() [884] instead of GREATEST() [784].

- *expr* IN (*value*,...) [784]

  Returns 1 if *expr* is equal to any of the values in the IN list, else returns 0. If all values are constants, they are evaluated according to the type of *expr* and sorted. The search for the item then is done using

a binary search. This means `IN` is very quick if the `IN` value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described in Section 11.2, “Type Conversion in Expression Evaluation”, but applied to all the arguments.

```
mysql> SELECT 2 IN (0,3,5,7);
        -> 0
mysql> SELECT 'wefwf' IN ('wee','wefwf','weg');
        -> 1
```

You should never mix quoted and unquoted values in an `IN` list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an `IN` expression like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

Instead, write it like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

The number of values in the `IN` list is only limited by the `max_allowed_packet` [418] value.

To comply with the SQL standard, from MySQL 4.1.0 on `IN` returns `NULL` not only if the expression on the left hand side is `NULL`, but also if no match is found in the list and one of the expressions in the list is `NULL`.

From MySQL 4.1.0 on, `IN()` syntax can also be used to write certain types of subqueries. See Section 12.2.8.3, “Subqueries with ANY, IN, or SOME”.

- *`expr`* `NOT IN (`*`value`*`,...)` [785]

  This is the same as `NOT (`*`expr`* `IN (`*`value`*`,...))`.

- `ISNULL(`*`expr`*`)` [785]

  If *`expr`* is `NULL`, `ISNULL()` [785] returns `1`, otherwise it returns `0`.

  ```
  mysql> SELECT ISNULL(1+1);
          -> 0
  mysql> SELECT ISNULL(1/0);
          -> 1
  ```

  `ISNULL()` [785] can be used instead of `=` [782] to test whether a value is `NULL`. (Comparing a value to `NULL` using `=` [782] always yields false.)

  The `ISNULL()` [785] function shares some special behaviors with the `IS NULL` [783] comparison operator. See the description of `IS NULL` [783].

- `INTERVAL(`*`N`*`,`*`N1`*`,`*`N2`*`,`*`N3`*`,...)` [785]

  Returns `0` if *`N`* < *`N1`*, `1` if *`N`* < *`N2`* and so on or `-1` if *`N`* is `NULL`. All arguments are treated as integers. It is required that *`N1`* < *`N2`* < *`N3`* < `...` < *`Nn`* for this function to work correctly. This is because a binary search is used (very fast).

  ```
  mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
          -> 3
  mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
  ```

```
        -> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

- `LEAST(value1,value2,...)` [786]

  With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

  - If the return value is used in an `INTEGER` context or all arguments are integer-valued, they are compared as integers.
  - If the return value is used in a `REAL` context or all arguments are real-valued, they are compared as reals.
  - If the arguments comprise a mix of numbers and strings, they are compared as numbers.
  - If any argument is a nonbinary (character) string, the arguments are compared as nonbinary strings.
  - In all other cases, the arguments are compared as binary strings.

  `LEAST()` [786] returns `NULL` only if all arguments are `NULL`.

```
mysql> SELECT LEAST(2,0);
        -> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
        -> 3.0
mysql> SELECT LEAST('B','A','C');
        -> 'A'
```

  Before MySQL 3.22.5, you can use `MIN()` [884] instead of `LEAST()` [786].

  Note that the preceding conversion rules can produce strange results in some borderline cases:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
        -> -9223372036854775808
```

  This happens because MySQL reads `9223372036854775808.0` in an integer context. The integer representation is not good enough to hold the value, so it wraps to a signed integer.

11.3.3 Logical Operators

Table 11.4 Logical Operators

Name	Description
`AND`, `&&` [787]	Logical AND
`NOT`, `!` [787]	Negates value
`||`, `OR` [787]	Logical OR
`XOR` [788]	Logical XOR

In SQL, all logical operators evaluate to `TRUE`, `FALSE`, or `NULL` (`UNKNOWN`). In MySQL, these are implemented as 1 (`TRUE`), 0 (`FALSE`), and `NULL`. Most of this is common to different SQL database servers, although some servers may return any nonzero value for `TRUE`.

MySQL evaluates any nonzero, non-`NULL` value to `TRUE`. For example, the following statements all assess to `TRUE`:

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- `NOT` [787], `!` [787]

  Logical NOT. Evaluates to `1` if the operand is `0`, to `0` if the operand is nonzero, and `NOT NULL` returns `NULL`.

  ```
  mysql> SELECT NOT 10;
          -> 0
  mysql> SELECT NOT 0;
          -> 1
  mysql> SELECT NOT NULL;
          -> NULL
  mysql> SELECT ! (1+1);
          -> 0
  mysql> SELECT ! 1+1;
          -> 1
  ```

  The last example produces `1` because the expression evaluates the same way as `(!1)+1`.

- `AND` [787], `&&` [787]

  Logical AND. Evaluates to `1` if all operands are nonzero and not `NULL`, to `0` if one or more operands are `0`, otherwise `NULL` is returned.

  ```
  mysql> SELECT 1 && 1;
          -> 1
  mysql> SELECT 1 && 0;
          -> 0
  mysql> SELECT 1 && NULL;
          -> NULL
  mysql> SELECT 0 && NULL;
          -> 0
  mysql> SELECT NULL && 0;
          -> 0
  ```

  Please note that MySQL versions prior to 4.0.5 stop evaluation when a `NULL` is encountered, rather than continuing the process to check for possible `0` values. This means that in these versions, `SELECT (NULL AND 0)` returns `NULL` instead of `0`. As of MySQL 4.0.5, the code has been re-engineered so that the result is always as prescribed by the SQL standards while still using the optimization wherever possible.

- `OR` [787], `||` [787]

  Logical OR. When both operands are non-`NULL`, the result is `1` if any operand is nonzero, and `0` otherwise. With a `NULL` operand, the result is `1` if the other operand is nonzero, and `NULL` otherwise. If both operands are `NULL`, the result is `NULL`.

  ```
  mysql> SELECT 1 || 1;
          -> 1
  mysql> SELECT 1 || 0;
          -> 1
  mysql> SELECT 0 || 0;
          -> 0
  ```

```
mysql> SELECT 0 || NULL;
        -> NULL
mysql> SELECT 1 || NULL;
        -> 1
```

- XOR [788]

  Logical XOR. Returns NULL if either operand is NULL. For non-NULL operands, evaluates to 1 if an odd number of operands is nonzero, otherwise 0 is returned.

```
mysql> SELECT 1 XOR 1;
        -> 0
mysql> SELECT 1 XOR 0;
        -> 1
mysql> SELECT 1 XOR NULL;
        -> NULL
mysql> SELECT 1 XOR 1 XOR 1;
        -> 1
```

  a XOR b is mathematically equal to (a AND (NOT b)) OR ((NOT a) and b).

  XOR [788] was added in MySQL 4.0.2.

11.3.4 Assignment Operators

Table 11.5 Assignment Operators

Name	Description
= [789]	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
:= [788]	Assign a value

- := [788]

  Assignment operator. Causes the user variable on the left hand side of the operator to take on the value to its right. The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same SET statement. You can perform multiple assignments in the same statement-

  Unlike = [789], the := [788] operator is never interpreted as a comparison operator. This means you can use := [788] in any valid SQL statement (not just in SET statements) to assign a value to a variable.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1

mysql> SELECT @var1:=COUNT(*) FROM t1;
        -> 4
mysql> SELECT @var1;
```

```
        -> 4
```

You can make value assignments using := [788] in other statements besides SELECT, such as UPDATE, as shown here:

```
mysql> SELECT @var1;
        -> 4
mysql> SELECT * FROM t1;
        -> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @var1;
        -> 1
mysql> SELECT * FROM t1;
        -> 2, 3, 5, 7
```

While it is also possible both to set and to read the value of the same variable in a single SQL statement using the := [788] operator, this is not recommended. Section 8.4, “User-Defined Variables”, explains why you should avoid doing this.

- = [789]

  This operator is used to perform value assignments in two cases, described in the next two paragraphs.

  Within a SET statement, = is treated as an assignment operator that causes the user variable on the left hand side of the operator to take on the value to its right. (In other words, when used in a SET statement, = is treated identically to := [788].) The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same SET statement.

  In the SET clause of an UPDATE statement, = also acts as an assignment operator; in this case, however, it causes the column named on the left hand side of the operator to assume the value given to the right, provided any WHERE conditions that are part of the UPDATE are met. You can make multiple assignments in the same SET clause of an UPDATE statement.

  In any other context, = is treated as a comparison operator [782].

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1
```

For more information, see Section 12.4.4, “SET Syntax”, Section 12.2.9, “UPDATE Syntax”, and Section 12.2.8, “Subquery Syntax”.

11.4 Control Flow Functions

Table 11.6 Flow Control Operators

Name	Description
CASE [790]	Case operator
IF() [790]	If/else construct
IFNULL() [791]	Null if/else construct
NULLIF() [791]	Return NULL if expr1 = expr2

- `CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN result ...] [ELSE result] END` [790]

  `CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END` [790]

  The first version returns the *result* where *value=compare_value*. The second version returns the result for the first condition that is true. If there was no matching result value, the result after ELSE is returned, or NULL if there is no ELSE part.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
    ->     WHEN 2 THEN 'two' ELSE 'more' END;
        -> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
        -> 'true'
mysql> SELECT CASE BINARY 'B'
    ->     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
        -> NULL
```

  Before MySQL 4.1, the type of the return value (INTEGER, DOUBLE, or STRING) is the same as the type of the first returned value (the expression after the first THEN). From MySQL 4.1.0, the default return type is the compatible aggregated type of all return values.

  Note that CASE [790] evaluation depends also on the context in which it is used. If used in string context, the result is returned as a string. If used in numeric context, the result is returned decimal, real, or integer value.

  CASE [790] was added in MySQL 3.23.3.

- `IF(expr1,expr2,expr3)` [790]

  If *expr1* is TRUE (*expr1 <> 0* and *expr1 <> NULL*) then IF() [790] returns *expr2*; otherwise it returns *expr3*. IF() [790] returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IF(1>2,2,3);
        -> 3
mysql> SELECT IF(1<2,'yes','no');
        -> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
        -> 'no'
```

  If only one of *expr2* or *expr3* is explicitly NULL, the result type of the IF() [790] function is the type of non-NULL expression. (This behavior was implemented in MySQL 4.0.3.)

  expr1 is evaluated as an integer value, which means that if you are testing floating-point or string values, you should do so using a comparison operation.

```
mysql> SELECT IF(0.1,1,0);
        -> 0
mysql> SELECT IF(0.1<>0,1,0);
        -> 1
```

In the first case shown, `IF(0.1)` [790] returns `0` because `0.1` is converted to an integer value, resulting in a test of `IF(0)` [790]. This may not be what you expect. In the second case, the comparison tests the original floating-point value to see whether it is nonzero. The result of the comparison is used as an integer.

The default return type of `IF()` [790] (which may matter when it is stored into a temporary table) is calculated in MySQL 3.23 as follows.

Expression	Return Value
`expr2` or *`expr3`* returns a string	string
`expr2` or *`expr3`* returns a floating-point value	floating-point
`expr2` or *`expr3`* returns an integer	integer

If *`expr2`* and *`expr3`* are both strings, the result is case sensitive if either string is case sensitive (starting from MySQL 3.23.51).

- `IFNULL(`*`expr1`*`,`*`expr2`*`)` [791]

If *`expr1`* is not `NULL`, `IFNULL()` [791] returns *`expr1`*; otherwise it returns *`expr2`*. `IFNULL()` [791] returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IFNULL(1,0);
        -> 1
mysql> SELECT IFNULL(NULL,10);
        -> 10
mysql> SELECT IFNULL(1/0,10);
        -> 10
mysql> SELECT IFNULL(1/0,'yes');
        -> 'yes'
```

In MySQL 4.0.6 and above, the default result value of `IFNULL(`*`expr1`*`,`*`expr2`*`)` [791] is the more "general" of the two expressions, in the order `STRING`, `REAL`, or `INTEGER`. The difference from earlier MySQL versions is mostly notable when you create a table based on expressions or MySQL has to internally store a value from `IFNULL()` [791] in a temporary table.

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| test  | varbinary(4) | NO   |     |         |       |
+-------+-------------+------+-----+---------+-------+
```

In MySQL 4.0, the type for the `test` column is `CHAR(4)`. In earlier versions, the type would be `BIGINT`.

- `NULLIF(`*`expr1`*`,`*`expr2`*`)` [791]

Returns `NULL` if *`expr1 = expr2`* is true, otherwise returns *`expr1`*. This is the same as `CASE WHEN` *`expr1 = expr2`* `THEN NULL ELSE` *`expr1`* `END` [790].

```
mysql> SELECT NULLIF(1,1);
        -> NULL
mysql> SELECT NULLIF(1,2);
        -> 1
```

Note that MySQL evaluates *expr1* twice if the arguments are not equal.

`NULLIF()` [791] was added in MySQL 3.23.15.

11.5 String Functions

Table 11.7 String Operators

Name	Description
`ASCII()` [793]	Return numeric value of left-most character
`BIN()` [794]	Return a string containing binary representation of a number
`BIT_LENGTH()` [794]	Return length of argument in bits
`CHAR_LENGTH()` [794]	Return number of characters in argument
`CHAR()` [794]	Return the character for each integer passed
`CHARACTER_LENGTH()` [794]	Synonym for CHAR_LENGTH()
`CONCAT_WS()` [795]	Return concatenate with separator
`CONCAT()` [795]	Return concatenated string
`ELT()` [795]	Return string at index number
`EXPORT_SET()` [795]	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
`FIELD()` [796]	Return the index (position) of the first argument in the subsequent arguments
`FIND_IN_SET()` [796]	Return the index position of the first argument within the second argument
`FORMAT()` [796]	Return a number formatted to specified number of decimal places
`HEX()` [796]	Return a hexadecimal representation of a decimal or string value
`INSERT()` [797]	Insert a substring at the specified position up to the specified number of characters
`INSTR()` [797]	Return the index of the first occurrence of substring
`LCASE()` [797]	Synonym for LOWER()
`LEFT()` [797]	Return the leftmost number of characters as specified
`LENGTH()` [797]	Return the length of a string in bytes
`LIKE` [804]	Simple pattern matching
`LOAD_FILE()` [798]	Load the named file
`LOCATE()` [798]	Return the position of the first occurrence of substring
`LOWER()` [798]	Return the argument in lowercase
`LPAD()` [798]	Return the string argument, left-padded with the specified string

Name	Description
LTRIM() [799]	Remove leading spaces
MAKE_SET() [799]	Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH [845]	Perform full-text search
MID() [799]	Return a substring starting from the specified position
NOT LIKE [806]	Negation of simple pattern matching
NOT REGEXP [808]	Negation of REGEXP
OCT() [821]	Return a string containing octal representation of a number
OCTET_LENGTH() [799]	Synonym for LENGTH()
ORD() [799]	Return character code for leftmost character of the argument
POSITION() [799]	Synonym for LOCATE()
QUOTE() [800]	Escape the argument for use in an SQL statement
REGEXP [808]	Pattern matching using regular expressions
REPEAT() [800]	Repeat a string the specified number of times
REPLACE() [800]	Replace occurrences of a specified string
REVERSE() [800]	Reverse the characters in a string
RIGHT() [800]	Return the specified rightmost number of characters
RLIKE [808]	Synonym for REGEXP
RPAD() [800]	Append string the specified number of times
RTRIM() [801]	Remove trailing spaces
SOUNDEX() [801]	Return a soundex string
SOUNDS LIKE [801]	Compare sounds
SPACE() [801]	Return a string of the specified number of spaces
STRCMP() [807]	Compare two strings
SUBSTR() [802]	Return the substring as specified
SUBSTRING_INDEX() [802]	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING() [802]	Return the substring as specified
TRIM() [803]	Remove leading and trailing spaces
UCASE() [803]	Synonym for UPPER()
UNHEX() [803]	Return a string containing hex representation of a number
UPPER() [804]	Convert to uppercase

String-valued functions return NULL if the length of the result would be greater than the value of the max_allowed_packet [418] system variable. See Section 7.8.2, “Tuning Server Parameters”.

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

- ASCII(*str*) [793]

Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns NULL if *str* is NULL. ASCII() [793] works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

See also the ORD() [799] function.

- BIN(*N*) [794]

  Returns a string representation of the binary value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,2) [818]. Returns NULL if *N* is NULL.

  ```
  mysql> SELECT BIN(12);
          -> '1100'
  ```

- BIT_LENGTH(*str*) [794]

  Returns the length of the string *str* in bits.

  ```
  mysql> SELECT BIT_LENGTH('text');
          -> 32
  ```

  BIT_LENGTH() [794] was added in MySQL 4.0.2.

- CHAR(*N*,... [USING *charset_name*]) [794]

  CHAR() [794] interprets each argument *N* as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.

  ```
  mysql> SELECT CHAR(77,121,83,81,'76');
          -> 'MySQL'
  mysql> SELECT CHAR(77,77.3,'77.3');
          -> 'MMM'
  ```

  CHAR() [794] returns a string in the connection character set. As of MySQL 4.1.16, the optional USING clause may be used to produce a string in a given character set:

  ```
  mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
  +--------------------+------------------------------+
  | CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
  +--------------------+------------------------------+
  | latin1              | utf8                         |
  +--------------------+------------------------------+
  ```

- CHAR_LENGTH(*str*) [794]

  Returns the length of the string *str*, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, LENGTH() [797] returns 10, whereas CHAR_LENGTH() [794] returns 5.

- CHARACTER_LENGTH(*str*) [794]

`CHARACTER_LENGTH()` [794] is a synonym for `CHAR_LENGTH()` [794].

- `CONCAT(str1,str2,...)` [795]

  Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

  ```
  SELECT CONCAT(CAST(int_col AS CHAR), char_col);
  ```

  `CONCAT()` [795] returns `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT CONCAT('My', 'S', 'QL');
         -> 'MySQL'
  mysql> SELECT CONCAT('My', NULL, 'QL');
         -> NULL
  mysql> SELECT CONCAT(14.3);
         -> '14.3'
  ```

  For quoted strings, concatenation can be performed by placing the strings next to each other:

  ```
  mysql> SELECT 'My' 'S' 'QL';
         -> 'MySQL'
  ```

- `CONCAT_WS(separator,str1,str2,...)` [795]

  `CONCAT_WS()` [795] stands for Concatenate With Separator and is a special form of `CONCAT()` [795]. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

  ```
  mysql> SELECT CONCAT_WS(',','First name','Second name','Last Name');
         -> 'First name,Second name,Last Name'
  mysql> SELECT CONCAT_WS(',','First name',NULL,'Last Name');
         -> 'First name,Last Name'
  ```

  `CONCAT_WS()` [795] skips any `NULL` values after the separator argument. Before MySQL 4.0.14, `CONCAT_WS()` [795] skips empty strings as well as `NULL` values.

- `ELT(N,str1,str2,str3,...)` [795]

  `ELT()` [795] returns the *N*th element of the list of strings: *str1* if *N* = `1`, *str2* if *N* = `2`, and so on. Returns `NULL` if *N* is less than `1` or greater than the number of arguments. `ELT()` [795] is the complement of `FIELD()` [796].

  ```
  mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
         -> 'ej'
  mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
         -> 'foo'
  ```

- `EXPORT_SET(bits,on,off[,separator[,number_of_bits]])` [795]

  Returns a string such that for every bit set in the value *bits*, you get an *on* string and for every bit not set in the value, you get an *off* string. Bits in *bits* are examined from right to left (from low-order to

high-order bits). Strings are added to the result from left to right, separated by the *separator* string (the default being the comma character ","). The number of bits examined is given by *number_of_bits* (defaults to 64).

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
        -> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6,'1','0',',',10);
        -> '0,1,1,0,0,0,0,0,0,0'
```

- FIELD(*str*,*str1*,*str2*,*str3*,...) [796]

  Returns the index (position) of *str* in the *str1*, *str2*, *str3*, ... list. Returns 0 if *str* is not found.

  If all arguments to FIELD() [796] are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

  If *str* is NULL, the return value is 0 because NULL fails equality comparison with any value. FIELD() [796] is the complement of ELT() [795].

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 0
```

- FIND_IN_SET(*str*,*strlist*) [796]

  Returns a value in the range of 1 to *N* if the string *str* is in the string list *strlist* consisting of *N* substrings. A string list is a string composed of substrings separated by "," characters. If the first argument is a constant string and the second is a column of type SET, the FIND_IN_SET() [796] function is optimized to use bit arithmetic. Returns 0 if *str* is not in *strlist* or if *strlist* is the empty string. Returns NULL if either argument is NULL. This function does not work properly if the first argument contains a comma (",") character.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

- FORMAT(*X*,*D*) [796]

  Formats the number *X* to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. If *D* is 0, the result has no decimal point or fractional part. *D* should be a constant value.

```
mysql> SELECT FORMAT(12332.123456, 4);
        -> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
        -> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
        -> '12,332'
```

- HEX(*str*) [796], HEX(*N*) [796]

  For a string argument *str* (supported as of MySQL 4.0.1), HEX() [796] returns a hexadecimal string representation of *str* where each byte of each character in *str* is converted to two hexadecimal digits. (Multi-byte characters therefore become more than two digits.) The inverse of this operation is performed by the UNHEX() [803] function.

For a numeric argument *N*, HEX() [796] returns a hexadecimal string representation of the value of *N* treated as a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,16) [818]. The inverse of this operation is performed by CONV(HEX(*N*),16,10) [818].

```
mysql> SELECT 0x616263, HEX('abc'), UNHEX(HEX('abc'));
        -> 'abc', 616263, 'abc'
mysql> SELECT HEX(255), CONV(HEX(255),16,10);
        -> 'FF', 255
```

- INSERT(*str*,*pos*,*len*,*newstr*) [797]

  Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns NULL if any argument is NULL.

  ```
  mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
          -> 'QuWhattic'
  mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
          -> 'Quadratic'
  mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
          -> 'QuWhat'
  ```

  This function is multi-byte safe.

- INSTR(*str*,*substr*) [797]

  Returns the position of the first occurrence of substring *substr* in string *str*. This is the same as the two-argument form of LOCATE() [798], except that the order of the arguments is reversed.

  ```
  mysql> SELECT INSTR('foobarbar', 'bar');
          -> 4
  mysql> SELECT INSTR('xbar', 'foobar');
          -> 0
  ```

  This function is multi-byte safe. In MySQL 3.23, this function is case sensitive. For 4.0 on, it is case sensitive only if either argument is a binary string.

- LCASE(*str*) [797]

  LCASE() [797] is a synonym for LOWER() [798].

- LEFT(*str*,*len*) [797]

  Returns the leftmost *len* characters from the string *str*.

  ```
  mysql> SELECT LEFT('foobarbar', 5);
          -> 'fooba'
  ```

- LENGTH(*str*) [797]

  Returns the length of the string *str*, measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, LENGTH() [797] returns 10, whereas CHAR_LENGTH() [794] returns 5.

  ```
  mysql> SELECT LENGTH('text');
  ```

```
        -> 4
```

- `LOAD_FILE(file_name)` [798]

  Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the `FILE` [491] privilege. The file must be readable by all and its size less than `max_allowed_packet` [418] bytes.

  If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

```
mysql> UPDATE t
           SET blob_col=LOAD_FILE('/tmp/picture')
           WHERE id=1;
```

  Before MySQL 3.23, you must read the file inside your application and create an `INSERT` statement to update the database with the file contents. If you are using the MySQL++ library, one way to do this can be found in the MySQL++ manual, available at http://tangentsoft.net/mysql++/doc/.

- `LOCATE(substr,str)` [798], `LOCATE(substr,str,pos)` [798]

  The first syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*. The second syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*, starting at position *`pos`*. Returns `0` if *`substr`* is not in *`str`*.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
        -> 7
```

  This function is multi-byte safe. In MySQL 3.23, this function is case sensitive. For 4.0 on, it is case sensitive only if either argument is a binary string.

- `LOWER(str)` [798]

  Returns the string *`str`* with all characters changed to lowercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

  `LOWER()` [798] (and `UPPER()` [804]) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

  This function is multi-byte safe.

- `LPAD(str,len,padstr)` [798]

Returns the string *str*, left-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
mysql> SELECT LPAD('hi',4,'??');
        -> '??hi'
mysql> SELECT LPAD('hi',1,'??');
        -> 'h'
```

- `LTRIM(str)` [799]

  Returns the string *str* with leading space characters removed.

```
mysql> SELECT LTRIM('  barbar');
        -> 'barbar'
```

  This function is multi-byte safe.

- `MAKE_SET(bits,str1,str2,...)` [799]

  Returns a set value (a string containing substrings separated by "`,`" characters) consisting of the strings that have the corresponding bit in *bits* set. *str1* corresponds to bit 0, *str2* to bit 1, and so on. `NULL` values in *str1*, *str2*, `...` are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
        -> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

- `MID(str,pos,len)` [799]

  `MID(str,pos,len)` [799] is a synonym for `SUBSTRING(str,pos,len)` [802].

- `OCTET_LENGTH(str)` [799]

  `OCTET_LENGTH()` [799] is a synonym for `LENGTH()` [797].

- `ORD(str)` [799]

  If the leftmost character of the string *str* is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

  If the leftmost character is not a multi-byte character, `ORD()` [799] returns the same value as the `ASCII()` [793] function.

```
mysql> SELECT ORD('2');
        -> 50
```

- `POSITION(substr IN str)` [799]

`POSITION(substr IN str)` [799] is a synonym for `LOCATE(substr,str)` [798].

- `QUOTE(str)` [800]

  Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of single quote (“`'`”), backslash (“`\`”), ASCII `NUL`, and Control-Z preceded by a backslash. If the argument is `NULL`, the return value is the word “NULL” without enclosing single quotation marks. The `QUOTE()` [800] function was added in MySQL 4.0.3.

```
mysql> SELECT QUOTE('Don\'t!');
        -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
        -> NULL
```

- `REPEAT(str,count)` [800]

  Returns a string consisting of the string *`str`* repeated *`count`* times. If *`count`* is less than 1, returns an empty string. Returns `NULL` if *`str`* or *`count`* are `NULL`.

```
mysql> SELECT REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

- `REPLACE(str,from_str,to_str)` [800]

  Returns the string *`str`* with all occurrences of the string *`from_str`* replaced by the string *`to_str`*. `REPLACE()` [800] performs a case-sensitive match when searching for *`from_str`*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

  This function is multi-byte safe.

- `REVERSE(str)` [800]

  Returns the string *`str`* with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

  This function is multi-byte safe.

- `RIGHT(str,len)` [800]

  Returns the rightmost *`len`* characters from the string *`str`*.

```
mysql> SELECT RIGHT('foobarbar', 4);
        -> 'rbar'
```

  This function is multi-byte safe.

- `RPAD(str,len,padstr)` [800]

  Returns the string *`str`*, right-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

```
mysql> SELECT RPAD('hi',5,'?');
        -> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
        -> 'h'
```

This function is multi-byte safe.

- `RTRIM(str)` [801]

  Returns the string *`str`* with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar   ');
        -> 'barbar'
```

  This function is multi-byte safe.

- `SOUNDEX(str)` [801]

  Returns a soundex string from *`str`*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` [801] function returns an arbitrarily long string. You can use `SUBSTRING()` [802] on the result to get a standard soundex string. All nonalphabetic characters in *`str`* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important

When using `SOUNDEX()` [801], you should be aware of the following limitations:

  - This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.
  - This function is not guaranteed to provide consistent results with strings that use multi-byte character sets, including `utf-8`.

    We hope to remove these limitations in a future release. See Bug #22638 for more information.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

Note

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- *`expr1`* `SOUNDS LIKE` *`expr2`* [801]

  This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)` [801]. It is available beginning with MySQL 4.1.0.

- `SPACE(N)` [801]

  Returns a string consisting of *`N`* space characters.

```
mysql> SELECT SPACE(6);
        -> '      '
```

The characters in the argument string must be legal hexadecimal digits: `'0'` .. `'9'`, `'A'` .. `'F'`, `'a'` .. `'f'`. If `UNHEX()` [803] encounters any nonhexadecimal digits in the argument, it returns `NULL`:

```
mysql> SELECT UNHEX('GG');
+-------------+
| UNHEX('GG') |
+-------------+
| NULL        |
+-------------+
```

A `NULL` result can occur if the argument to `UNHEX()` [803] is a `BINARY` column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example `'aa'` is stored into a `CHAR(3)` column as `'aa '` and retrieved as `'aa'` (with the trailing pad space stripped), so `UNHEX()` [803] for the column value returns `'A'`. By contrast `'aa'` is stored into a `BINARY(3)` column as `'aa\0'` and retrieved as `'aa\0'` (with the trailing pad `0x00` byte not stripped). `'\0'` is not a legal hexadecimal digit, so `UNHEX()` [803] for the column value returns `NULL`.

- `SUBSTR(str,pos)` [802], `SUBSTR(str FROM pos)` [802], `SUBSTR(str,pos,len)` [802], `SUBSTR(str FROM pos FOR len)` [802]

  `SUBSTR()` [802] is a synonym for `SUBSTRING()` [802]. It was added in MySQL 4.1.1.

- `SUBSTRING(str,pos)` [802], `SUBSTRING(str FROM pos)` [802], `SUBSTRING(str,pos,len)` [802], `SUBSTRING(str FROM pos FOR len)` [802]

  The forms without a *`len`* argument return a substring from string *`str`* starting at position *`pos`*. The forms with a *`len`* argument return a substring *`len`* characters long from string *`str`*, starting at position *`pos`*. The forms that use `FROM` are standard SQL syntax. Beginning with MySQL 4.1.0, it is possible to use a negative value for *`pos`*. In this case, the beginning of the substring is *`pos`* characters from the end of the string, rather than the beginning. A negative value may be used for *`pos`* in any of the forms of this function.

  For all forms of `SUBSTRING()` [802], the position of the first character in the string from which the substring is to be extracted is reckoned as `1`.

  ```
  mysql> SELECT SUBSTRING('Quadratically',5);
          -> 'ratically'
  mysql> SELECT SUBSTRING('foobarbar' FROM 4);
          -> 'barbar'
  mysql> SELECT SUBSTRING('Quadratically',5,6);
          -> 'ratica'
  mysql> SELECT SUBSTRING('Sakila', -3);
          -> 'ila'
  mysql> SELECT SUBSTRING('Sakila', -5, 3);
          -> 'aki'
  mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
          -> 'ki'
  ```

  This function is multi-byte safe.

  If *`len`* is less than 1, the result is the empty string.

- `SUBSTRING_INDEX(str,delim,count)` [802]

Returns the substring from string *str* before *count* occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned. SUBSTRING_INDEX() [802] performs a case-sensitive match when searching for *delim*.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
        -> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
        -> 'mysql.com'
```

This function is multi-byte safe.

- TRIM([{BOTH | LEADING | TRAILING} [*remstr*] FROM] *str*) [803], TRIM([*remstr* FROM] *str*) [803]

  Returns the string *str* with all *remstr* prefixes or suffixes removed. If none of the specifiers BOTH, LEADING, or TRAILING is given, BOTH is assumed. *remstr* is optional and, if not specified, spaces are removed.

```
mysql> SELECT TRIM('  bar   ');
        -> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

  This function is multi-byte safe.

- UCASE(*str*) [803]

  UCASE() [803] is a synonym for UPPER() [804].

- UNHEX(*str*) [803]

  For a string argument *str*, UNHEX(*str*) [803] interprets each pair of characters in the argument as a hexadecimal number and converts it to the byte represented by the number. The return value is a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
        -> 'MySQL'
mysql> SELECT 0x4D7953514C;
        -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
        -> 'string'
mysql> SELECT HEX(UNHEX('1267'));
        -> '1267'
```

  The characters in the argument string must be legal hexadecimal digits: '0' .. '9', 'A' .. 'F', 'a' .. 'f'. If the argument contains any nonhexadecimal digits, the result is NULL:

```
mysql> SELECT UNHEX('GG');
+-------------+
| UNHEX('GG') |
+-------------+
| NULL        |
```

```
+------------+
```

For a numeric argument *N*, the inverse of `HEX(N)` [796] is not performed by `UNHEX()` [803]. Use `CONV(HEX(N),16,10)` [818] instead. See the description of `HEX()` [796].

`UNHEX()` [803] was added in MySQL 4.1.2.

- `UPPER(str)` [804]

  Returns the string *str* with all characters changed to uppercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT UPPER('Hej');
        -> 'HEJ'
```

`UPPER()` [804] is ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). The description of `LOWER()` [798] shows how to perform lettercase conversion of binary strings.

This function is multi-byte safe.

11.5.1 String Comparison Functions

Table 11.8 String Comparison Operators

Name	Description
`LIKE` [804]	Simple pattern matching
`NOT LIKE` [806]	Negation of simple pattern matching
`STRCMP()` [807]	Compare two strings

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- *expr* `LIKE` *pat* `[ESCAPE '`*escape_char*`']` [804]

  Pattern matching using SQL simple regular expression comparison. Returns `1` (`TRUE`) or `0` (`FALSE`). If either *expr* or *pat* is `NULL`, the result is `NULL`.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Per the SQL standard, `LIKE` [804] performs matching on a per-character basis, thus it can produce results different from the `=` [782] comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----------------------------------------+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-------------------------------------+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

In particular, trailing spaces are significant, which is not true for CHAR or VARCHAR comparisons performed with the = [782] operator:

```
mysql> SELECT 'a' = 'a ', 'a' LIKE 'a ';
+-----------+--------------+
| 'a' = 'a ' | 'a' LIKE 'a ' |
+-----------+--------------+
|          1 |             0 |
+-----------+--------------+
1 row in set (0.00 sec)
```

With LIKE [804] you can use the following two wildcard characters in the pattern.

Character	Description
%	Matches any number of characters, even zero characters
_	Matches exactly one character

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the ESCAPE character, "\" is assumed.

String	Description
\%	Matches one "%" character
_	Matches one "_" character

```
mysql> SELECT 'David!' LIKE 'David\_';
        -> 0
mysql> SELECT 'David_' LIKE 'David\_';
        -> 1
```

To specify a different escape character, use the ESCAPE clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

The following two statements illustrate that string comparisons are not case sensitive unless one of the operands is a binary string:

```
mysql> SELECT 'abc' LIKE 'ABC';
        -> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
        -> 0
```

In MySQL, LIKE [804] is permitted on numeric expressions. (This is an extension to the standard SQL LIKE [804].)

```
mysql> SELECT 10 LIKE '1%';
        -> 1
```

Note

Because MySQL uses C escape syntax in strings (for example, "\n" to represent a newline character), you must double any "\" that you use in LIKE [804] strings. For example, to search for "\n", specify it as "\\n". To search for "\", specify it as "\\\\"; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against.

Exception: At the end of the pattern string, backslash can be specified as "\\". At the end of the string, backslash stands for itself because there is nothing following to escape. Suppose that a table contains the following values:

```
mysql> SELECT filename FROM t1;
+--------------+
| filename     |
+--------------+
| C:           |
| C:\          |
| C:\Programs  |
| C:\Programs\ |
+--------------+
```

To test for values that end with backslash, you can match the values using either of the following patterns:

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+--------------+---------------------+
| filename     | filename LIKE '%\\' |
+--------------+---------------------+
| C:           |                   0 |
| C:\          |                   1 |
| C:\Programs  |                   0 |
| C:\Programs\ |                   1 |
+--------------+---------------------+

mysql> SELECT filename, filename LIKE '%\\\\' FROM t1;
+--------------+-----------------------+
| filename     | filename LIKE '%\\\\' |
+--------------+-----------------------+
| C:           |                     0 |
| C:\          |                     1 |
| C:\Programs  |                     0 |
| C:\Programs\ |                     1 |
+--------------+-----------------------+
```

- *expr* NOT LIKE *pat* [ESCAPE '*escape_char*'] [806]

  This is the same as NOT (*expr* LIKE *pat* [ESCAPE '*escape_char*']).

Note

Aggregate queries involving NOT LIKE [806] comparisons with columns containing NULL may yield unexpected results. For example, consider the following table and data:

```
CREATE TABLE foo (bar VARCHAR(10));
```

```
INSERT INTO foo VALUES (NULL), (NULL);
```

The query `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';` returns `0`. You might assume that `SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%';` would return `2`. However, this is not the case: The second query returns `0`. This is because `NULL NOT LIKE expr` always returns `NULL`, regardless of the value of *`expr`*. The same is true for aggregate queries involving `NULL` and comparisons using `NOT RLIKE` [808] or `NOT REGEXP` [808]. In such cases, you must test explicitly for `NOT NULL` using `OR` [787] (and not `AND` [787]), as shown here:

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- `STRCMP(expr1,expr2)` [807]

  `STRCMP()` [807] returns `0` if the strings are the same, `-1` if the first argument is smaller than the second according to the current sort order, and `1` otherwise.

```
mysql> SELECT STRCMP('text', 'text2');
        -> -1
mysql> SELECT STRCMP('text2', 'text');
        -> 1
mysql> SELECT STRCMP('text', 'text');
        -> 0
```

  As of MySQL 4.1, `STRCMP()` [807] performs the comparison using the collation of the arguments.

```
mysql> SET @s1 = _latin1 'x' COLLATE latin1_general_ci;
mysql> SET @s2 = _latin1 'X' COLLATE latin1_general_ci;
mysql> SET @s3 = _latin1 'x' COLLATE latin1_general_cs;
mysql> SET @s4 = _latin1 'X' COLLATE latin1_general_cs;
mysql> SELECT STRCMP(@s1, @s2), STRCMP(@s3, @s4);
+------------------+------------------+
| STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |
+------------------+------------------+
|                0 |                1 |
+------------------+------------------+
```

  If the collations are incompatible, one of the arguments must be converted to be compatible with the other. See Section 9.1.7.5, “Collation of Expressions”.

```
mysql> SELECT STRCMP(@s1, @s3);
ERROR 1267 (HY000) at line 10: Illegal mix of collations (latin1_general_ci,IMPLICIT) and (latin1_genera
mysql> SELECT STRCMP(@s1, @s3 COLLATE latin1_general_ci);
+-------------------------------------------+
| STRCMP(@s1, @s3 COLLATE latin1_general_ci) |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
```

  In MySQL 4.0, `STRCMP()` [807] performs the comparison using the current character set. This makes the default comparison behavior case insensitive unless one or both of the operands are binary strings. Before MySQL 4.0, `STRCMP()` [807] is case sensitive.

11.5.2 Regular Expressions

Table 11.9 String Regular Expression Operators

Name	Description
NOT REGEXP [808]	Negation of REGEXP
REGEXP [808]	Pattern matching using regular expressions
RLIKE [808]	Synonym for REGEXP

A regular expression is a powerful way of specifying a pattern for a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. MySQL uses the extended version to support pattern-matching operations performed with the REGEXP [808] operator in SQL statements.

This section summarizes, with examples, the special characters and constructs that can be used in MySQL for REGEXP [808] operations. It does not contain all the details that can be found in Henry Spencer's `regex(7)` manual page. That manual page is included in MySQL source distributions, in the `regex.7` file under the `regex` directory. See also Section 3.3.4.7, “Pattern Matching”.

Regular Expression Operators

- *expr* NOT REGEXP *pat* [808], *expr* NOT RLIKE *pat* [808]

  This is the same as NOT (*expr* REGEXP *pat*).

- *expr* REGEXP *pat* [808], *expr* RLIKE *pat* [808]

  Performs a pattern match of a string expression *expr* against a pattern *pat*. The pattern can be an extended regular expression, the syntax for which is discussed later in this section. Returns 1 if *expr* matches *pat*; otherwise it returns 0. If either *expr* or *pat* is NULL, the result is NULL. RLIKE [808] is a synonym for REGEXP [808], provided for mSQL compatibility.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Note

  Because MySQL uses the C escape syntax in strings (for example, “\n” to represent the newline character), you must double any “\” that you use in your REGEXP [808] strings.

  As of MySQL 3.23.4, REGEXP [808] is not case sensitive, except when used with binary strings.

```
mysql> SELECT 'Monty!' REGEXP '.*';
        -> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
        -> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
        -> 1  0
mysql> SELECT 'a' REGEXP '^[a-d]';
        -> 1
```

  REGEXP [808] and RLIKE [808] use the current character set when deciding the type of a character. The default is `latin1` (cp1252 West European).

Warning

The `REGEXP` [808] and `RLIKE` [808] operators work in byte-wise fashion, so they are not multi-byte safe and may produce unexpected results with multi-byte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

Syntax of Regular Expressions

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|word` matches either the string `hello` or the string `word`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression for the `REGEXP` [808] operator may use any of the following special characters and constructs:

- `^`

  Match the beginning of a string.

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';                -> 0
mysql> SELECT 'fofo' REGEXP '^fo';                   -> 1
```

- `$`

  Match the end of a string.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';              -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';                 -> 0
```

- `.`

  Match any character (including carriage return and newline).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';                 -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';             -> 1
```

- `a*`

  Match any sequence of zero or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';                  -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';                -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';                   -> 1
```

- `a+`

  Match any sequence of one or more `a` characters.

```
```

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';                    -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';                     -> 0
```

- `a?`

  Match either zero or one `a` character.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';                     -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';                    -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';                   -> 0
```

- `de|abc`

  Match either of the sequences `de` or `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';                    -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';                   -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';                   -> 1
mysql> SELECT 'apa' REGEXP '^(pi|apa)$';               -> 1
mysql> SELECT 'pi' REGEXP '^(pi|apa)$';                -> 1
mysql> SELECT 'pix' REGEXP '^(pi|apa)$';               -> 0
```

- `(abc)*`

  Match zero or more instances of the sequence `abc`.

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';                   -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';                  -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';                 -> 1
```

- `{1}`, `{2,3}`

  `{n}` or `{m,n}` notation provides a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. `m` and `n` are integers.

  - `a*`

    Can be written as `a{0,}`.

  - `a+`

    Can be written as `a{1,}`.

  - `a?`

    Can be written as `a{0,1}`.

  To be more precise, `a{n}` matches exactly `n` instances of `a`. `a{n,}` matches `n` or more instances of `a`. `a{m,n}` matches `m` through `n` instances of `a`, inclusive.

  `m` and `n` must be in the range from `0` to `RE_DUP_MAX` (default 255), inclusive. If both `m` and `n` are given, `m` must be less than or equal to `n`.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';             -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';             -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';          -> 1
```

- `[a-dX]`, `[^a-dX]`

Matches any character that is (or is not, if ^ is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';                -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';              -> 0
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$';             -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';            -> 0
mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$';           -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$';          -> 0
```

- `[.characters.]`

  Within a bracket expression (written using `[` and `]`), matches the sequence of characters of that collating element. `characters` is either a single character or a character name like `newline`. The following table lists the permissible character names.

  The following table shows the permissible character names and the characters that they match. For characters given as numeric values, the values are represented in octal.

Name	Character	Name	Character
NUL	0	SOH	001
STX	002	ETX	003
EOT	004	ENQ	005
ACK	006	BEL	007
alert	007	BS	010
backspace	'\b'	HT	011
tab	'\t'	LF	012
newline	'\n'	VT	013
vertical-tab	'\v'	FF	014
form-feed	'\f'	CR	015
carriage-return	'\r'	SO	016
SI	017	DLE	020
DC1	021	DC2	022
DC3	023	DC4	024
NAK	025	SYN	026
ETB	027	CAN	030
EM	031	SUB	032
ESC	033	IS4	034
FS	034	IS3	035
GS	035	IS2	036
RS	036	IS1	037
US	037	space	' '

Name	Character	Name	Character
exclamation-mark	'!'	quotation-mark	'"'
number-sign	'#'	dollar-sign	'$'
percent-sign	'%'	ampersand	'&'
apostrophe	'\''	left-parenthesis	'('
right-parenthesis	')'	asterisk	'*'
plus-sign	'+'	comma	','
hyphen	'-'	hyphen-minus	'-'
period	'.'	full-stop	'.'
slash	'/'	solidus	'/'
zero	'0'	one	'1'
two	'2'	three	'3'
four	'4'	five	'5'
six	'6'	seven	'7'
eight	'8'	nine	'9'
colon	':'	semicolon	';'
less-than-sign	'<'	equals-sign	'='
greater-than-sign	'>'	question-mark	'?'
commercial-at	'@'	left-square-bracket	'['
backslash	'\\'	reverse-solidus	'\\'
right-square-bracket	']'	circumflex	'^'
circumflex-accent	'^'	underscore	'_'
low-line	'_'	grave-accent	'`'
left-brace	'{'	left-curly-bracket	'{'
vertical-line	'|'	right-brace	'}'
right-curly-bracket	'}'	tilde	'~'
DEL	177		

```
mysql> SELECT '~' REGEXP '[[.~.]]';                    -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';                 -> 1
```

- `[=character_class=]`

  Within a bracket expression (written using `[` and `]`), `[=character_class=]` represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if `o` and `(+)` are the members of an equivalence class, `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be used as an endpoint of a range.

- `[:character_class:]`

Within a bracket expression (written using `[` and `]`), `[:character_class:]` represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

Character Class Name	Meaning
`alnum`	Alphanumeric characters
`alpha`	Alphabetic characters
`blank`	Whitespace characters
`cntrl`	Control characters
`digit`	Digit characters
`graph`	Graphic characters
`lower`	Lowercase alphabetic characters
`print`	Graphic or space characters
`punct`	Punctuation characters
`space`	Space, tab, newline, and carriage return
`upper`	Uppercase alphabetic characters
`xdigit`	Hexadecimal digit characters

```
mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+';       -> 1
mysql> SELECT '!!' REGEXP '[[:alnum:]]+';               -> 0
```

- `[[:<:]]`, `[[:>:]]`

  These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the `alnum` class or an underscore (_).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';   -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';  -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';                       -> 0
mysql> SELECT '1+2' REGEXP '1\+2';                      -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                     -> 1
```

11.6 Numeric Functions and Operators

Table 11.10 Numeric Functions and Operators

Name	Description
ABS() [817]	Return the absolute value

Name	Description
ACOS() [817]	Return the arc cosine
ASIN() [818]	Return the arc sine
ATAN2(), ATAN() [818]	Return the arc tangent of the two arguments
ATAN() [818]	Return the arc tangent
CEIL() [818]	Return the smallest integer value not less than the argument
CEILING() [818]	Return the smallest integer value not less than the argument
CONV() [818]	Convert numbers between different number bases
COS() [819]	Return the cosine
COT() [819]	Return the cotangent
CRC32() [819]	Compute a cyclic redundancy check value
DEGREES() [819]	Convert radians to degrees
DIV [816]	Integer division
/ [816]	Division operator
EXP() [819]	Raise to the power of
FLOOR() [820]	Return the largest integer value not greater than the argument
LN() [820]	Return the natural logarithm of the argument
LOG10() [821]	Return the base-10 logarithm of the argument
LOG2() [821]	Return the base-2 logarithm of the argument
LOG() [820]	Return the natural logarithm of the first argument
- [815]	Minus operator
MOD() [821]	Return the remainder
% or MOD [816]	Modulo operator
PI() [822]	Return the value of pi
+ [815]	Addition operator
POW() [822]	Return the argument raised to the specified power
POWER() [822]	Return the argument raised to the specified power
RADIANS() [822]	Return argument converted to radians
RAND() [822]	Return a random floating-point value
ROUND() [823]	Round the argument
SIGN() [824]	Return the sign of the argument
SIN() [824]	Return the sine of the argument
SQRT() [824]	Return the square root of the argument
TAN() [824]	Return the tangent of the argument
* [816]	Multiplication operator
TRUNCATE() [825]	Truncate to specified number of decimal places
- [815]	Change the sign of the argument

11.6.1 Arithmetic Operators

Table 11.11 Arithmetic Operators

Name	Description
DIV [816]	Integer division
/ [816]	Division operator
- [815]	Minus operator
% or MOD [816]	Modulo operator
+ [815]	Addition operator
* [816]	Multiplication operator
- [815]	Change the sign of the argument

The usual arithmetic operators are available. The precision of the result is determined according to the following rules:

- In the case of - [815], + [815], and * [816], the result is calculated with BIGINT (64-bit) precision if both operands are integers.
- If both operands are integers and any of them are unsigned, the result is an unsigned integer. For subtraction, if the NO_UNSIGNED_SUBTRACTION [459] SQL mode is enabled, the result is signed even if any operand is unsigned.
- If any of the operands of a + [815], - [815], / [816], * [816], % [816] is a real or string value, the precision of the result is the precision of the operand with the maximum precision.

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, resolves first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.57692308`).

Because of these rules and the way they are applied, care should be taken to ensure that components and subcomponents of a calculation use the appropriate level of precision. See Section 11.10, “Cast Functions and Operators”.

For information about handling of overflow in numeric expression evaluation, see Section 10.2.5, “Out-of-Range and Overflow Handling”.

Arithmetic operators apply to numbers. For other types of values, alternative operations may be available. For example, to add date values, use DATE_ADD() [829]; see Section 11.7, “Date and Time Functions”.

- + [815]

  Addition:

```
mysql> SELECT 3+5;
        -> 8
```

- - [815]

  Subtraction:

```
mysql> SELECT 3-5;
        -> -2
```

- - [815]

  Unary minus. This operator changes the sign of the operand.

```
mysql> SELECT - 2;
        -> -2
```

Note

If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of -2^{63}.

- `*` [816]

  Multiplication:

```
mysql> SELECT 3*5;
        -> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
        -> 324518553658426726783156020576256.0
```

- `/` [816]

  Division:

```
mysql> SELECT 3/5;
        -> 0.60
```

  Division by zero produces a `NULL` result:

```
mysql> SELECT 102/(1-1);
        -> NULL
```

  A division is calculated with `BIGINT` arithmetic only if performed in a context where its result is converted to an integer.

- `DIV` [816]

  Integer division. Similar to `FLOOR()` [820], but is safe with `BIGINT` values. Incorrect results may occur for noninteger operands that exceed `BIGINT` range.

```
mysql> SELECT 5 DIV 2;
        -> 2
```

  `DIV` [816] was implemented in MySQL 4.1.0.

- *`N`* `%` *`M`* [816], *`N`* `MOD` *`M`* [816]

  Modulo operation. Returns the remainder of *`N`* divided by *`M`*. For more information, see the description for the `MOD()` [821] function in Section 11.6.2, “Mathematical Functions”.

11.6.2 Mathematical Functions

Table 11.12 Mathematical Functions

Name	Description
`ABS()` [817]	Return the absolute value
`ACOS()` [817]	Return the arc cosine

Name	Description
ASIN() [818]	Return the arc sine
ATAN2(), ATAN() [818]	Return the arc tangent of the two arguments
ATAN() [818]	Return the arc tangent
CEIL() [818]	Return the smallest integer value not less than the argument
CEILING() [818]	Return the smallest integer value not less than the argument
CONV() [818]	Convert numbers between different number bases
COS() [819]	Return the cosine
COT() [819]	Return the cotangent
CRC32() [819]	Compute a cyclic redundancy check value
DEGREES() [819]	Convert radians to degrees
EXP() [819]	Raise to the power of
FLOOR() [820]	Return the largest integer value not greater than the argument
LN() [820]	Return the natural logarithm of the argument
LOG10() [821]	Return the base-10 logarithm of the argument
LOG2() [821]	Return the base-2 logarithm of the argument
LOG() [820]	Return the natural logarithm of the first argument
MOD() [821]	Return the remainder
PI() [822]	Return the value of pi
POW() [822]	Return the argument raised to the specified power
POWER() [822]	Return the argument raised to the specified power
RADIANS() [822]	Return argument converted to radians
RAND() [822]	Return a random floating-point value
ROUND() [823]	Round the argument
SIGN() [824]	Return the sign of the argument
SIN() [824]	Return the sine of the argument
SQRT() [824]	Return the square root of the argument
TAN() [824]	Return the tangent of the argument
TRUNCATE() [825]	Truncate to specified number of decimal places

All mathematical functions return NULL in the event of an error.

- ABS(*X*) [817]

  Returns the absolute value of *X*.

```
mysql> SELECT ABS(2);
        -> 2
mysql> SELECT ABS(-32);
        -> 32
```

  This function is safe to use with BIGINT values.

- ACOS(*X*) [817]

Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns `NULL` if *X* is not in the range `-1` to `1`.

```
mysql> SELECT ACOS(1);
        -> 0.000000
mysql> SELECT ACOS(1.0001);
        -> NULL
mysql> SELECT ACOS(0);
        -> 1.570796
```

- `ASIN(X)` [818]

  Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns `NULL` if *X* is not in the range `-1` to `1`.

```
mysql> SELECT ASIN(0.2);
        -> 0.201358
mysql> SELECT ASIN('foo');
        -> 0.000000
```

- `ATAN(X)` [818]

  Returns the arc tangent of *X*, that is, the value whose tangent is *X*.

```
mysql> SELECT ATAN(2);
        -> 1.107149
mysql> SELECT ATAN(-2);
        -> -1.107149
```

- `ATAN(Y,X)` [818], `ATAN2(Y,X)` [818]

  Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of *`Y / X`*, except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2,2);
        -> -0.785398
mysql> SELECT ATAN2(PI(),0);
        -> 1.570796
```

- `CEIL(X)` [818]

  `CEIL()` [818] is a synonym for `CEILING()` [818]. It was added in MySQL 4.0.6.

- `CEILING(X)` [818]

  Returns the smallest integer value not less than *X*.

```
mysql> SELECT CEILING(1.23);
        -> 2
mysql> SELECT CEILING(-1.23);
        -> -1
```

  Note that the return value is converted to a `BIGINT`.

- `CONV(N,from_base,to_base)` [818]

  Converts numbers between different number bases. Returns a string representation of the number *N*, converted from base *`from_base`* to base *`to_base`*. Returns `NULL` if any argument is `NULL`. The

argument *N* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If *to_base* is a negative number, *N* is regarded as a signed number. Otherwise, *N* is treated as unsigned. CONV() [818] works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
        -> '1010'
mysql> SELECT CONV('6E',18,8);
        -> '172'
mysql> SELECT CONV(-17,10,-18);
        -> '-H'
mysql> SELECT CONV(10+'10'+'10'+0xa,10,10);
        -> '40'
```

- COS(*X*) [819]

  Returns the cosine of *X*, where *X* is given in radians.

```
mysql> SELECT COS(PI());
        -> -1.000000
```

- COT(*X*) [819]

  Returns the cotangent of *X*.

```
mysql> SELECT COT(12);
        -> -1.57267341
mysql> SELECT COT(0);
        -> NULL
```

- CRC32(*expr*) [819]

  Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is NULL if the argument is NULL. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
        -> 3259397556
```

  CRC32() [819] is available as of MySQL 4.1.0.

- DEGREES(*X*) [819]

  Returns the argument *X*, converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
        -> 180.000000
```

- EXP(*X*) [819]

  Returns the value of *e* (the base of natural logarithms) raised to the power of *X*. The inverse of this function is the LOG() [820]. In MySQL 4.0.3 or later, its inverse is LOG() [820] using a single argument or LN() [820].

```
mysql> SELECT EXP(2);
        -> 7.3890560989307
mysql> SELECT EXP(-2);
        -> 0.13533528323661
mysql> SELECT EXP(0);
```

```
        -> 1
```

- `FLOOR(X)` [820]

  Returns the largest integer value not greater than *X*.

```
mysql> SELECT FLOOR(1.23);
        -> 1
mysql> SELECT FLOOR(-1.23);
        -> -2
```

  Note that the return value is converted to a `BIGINT`.

- `FORMAT(X,D)` [796]

  Formats the number *X* to a format like `'#,###,###.##'`, rounded to *D* decimal places, and returns the result as a string. For details, see Section 11.5, “String Functions”.

- `HEX(N_or_S)` [796]

  This function can be used to obtain a hexadecimal representation of a decimal number or (beginning with MySQL 4.0.1) a string; the manner in which it does so varies according to the argument's type. See this function's description in Section 11.5, “String Functions”, for details.

- `LN(X)` [820]

  Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*. If *X* is less than or equal to 0, then `NULL` is returned.

```
mysql> SELECT LN(2);
        -> 0.69314718055995
mysql> SELECT LN(-2);
        -> NULL
```

  This function was added in MySQL 4.0.3. It is synonymous with `LOG(X)` [820]. The inverse of this function is the `EXP()` [819] function.

- `LOG(X)` [820], `LOG(B,X)` [820]

  If called with one parameter, this function returns the natural logarithm of *X*. If *X* is less than or equal to 0, then `NULL` is returned.

  The inverse of this function (when called with a single argument) is the `EXP()` [819] function.

```
mysql> SELECT LOG(2);
        -> 0.69314718055995
mysql> SELECT LOG(-2);
        -> NULL
```

  If called with two parameters, this function returns the logarithm of *X* to the base *B*. If *X* is less than or equal to 0, or if *B* is less than or equal to 1, then `NULL` is returned.

```
mysql> SELECT LOG(2,65536);
        -> 16.000000
mysql> SELECT LOG(10,100);
        -> 2
mysql> SELECT LOG(1,100);
        -> NULL
```

The arbitrary base option was added in MySQL 4.0.3. `LOG(B,X)` [820] is equivalent to `LOG(X) / LOG(B)` [820].

- `LOG2(X)` [821]

  Returns the base-2 logarithm of *X*.

```
mysql> SELECT LOG2(65536);
        -> 16.000000
mysql> SELECT LOG2(-100);
        -> NULL
```

  `LOG2()` [821] is useful for finding out how many bits a number would require for storage. This function was added in MySQL 4.0.3. In earlier versions, you can use `LOG(X) / LOG(2)` [820] instead.

- `LOG10(X)` [821]

  Returns the base-10 logarithm of *X*.

```
mysql> SELECT LOG10(2);
        -> 0.301030
mysql> SELECT LOG10(100);
        -> 2.000000
mysql> SELECT LOG10(-100);
        -> NULL
```

- `MOD(N,M)` [821], `N % M` [816], `N MOD M` [816]

  Modulo operation. Returns the remainder of *N* divided by *M*.

```
mysql> SELECT MOD(234, 10);
        -> 4
mysql> SELECT 253 % 7;
        -> 1
mysql> SELECT MOD(29,9);
        -> 2
mysql> SELECT 29 MOD 9;
        -> 2
```

  This function is safe to use with `BIGINT` values. The `N MOD M` syntax works only as of MySQL 4.1.0.

  As of MySQL 4.1.7, `MOD()` [821] works on values that have a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5,3);
        -> 1.5
```

  Before MySQL 4.1.7, `MOD()` [821] rounds arguments with a fractional value to integers and returns an integer result:

```
mysql> SELECT MOD(34.5,3);
        -> 2
```

  `MOD(N,0)` [821] returns `NULL`.

- `OCT(N)` [821]

Returns a string representation of the octal value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,8) [818]. Returns NULL if *N* is NULL.

```
mysql> SELECT OCT(12);
        -> '14'
```

- PI() [822]

  Returns the value of π (pi). The default number of decimal places displayed is five, but MySQL uses the full double-precision value internally.

```
mysql> SELECT PI();
        -> 3.141593
mysql> SELECT PI()+0.00000000000000000;
        -> 3.141592653589793116
```

- POW(*X*,*Y*) [822]

  Returns the value of *X* raised to the power of *Y*.

```
mysql> SELECT POW(2,2);
        -> 4.000000
mysql> SELECT POW(2,-2);
        -> 0.250000
```

- POWER(*X*,*Y*) [822]

  This is a synonym for POW() [822].

- RADIANS(*X*) [822]

  Returns the argument *X*, converted from degrees to radians. (Note that π radians equals 180 degrees.)

```
mysql> SELECT RADIANS(90);
        -> 1.570796
```

- RAND() [822], RAND(*N*) [822]

  Returns a random floating-point value *v* in the range 0 <= *v* < 1.0. If a constant integer argument *N* is specified, it is used as the seed value, which produces a repeatable sequence of column values. In the following example, note that the sequences of values produced by RAND(3) is the same both places where it occurs.

```
mysql> CREATE TABLE t (i INT);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.61914388706828 |
|    2 | 0.93845168309142 |
```

```
|    3 | 0.83482678498591 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.35877890638893 |
|    2 | 0.28941420772058 |
|    3 | 0.37073435016976 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.01 sec)
```

The effect of using a nonconstant argument is undefined. As of MySQL 4.1.15, nonconstant arguments are not permitted.

To obtain a random integer *R* in the range *i* <= *R* < *j*, use the expression `FLOOR(i + RAND() * (j` [820] `- i))`. For example, to obtain a random integer in the range the range `7` <= *R* < `12`, you could use the following statement:

```
SELECT FLOOR(7 + (RAND() * 5));
```

`RAND()` [822] in a `WHERE` clause is re-evaluated every time the `WHERE` is executed.

You cannot use a column with `RAND()` [822] values in an `ORDER BY` clause, because `ORDER BY` would evaluate the column multiple times. However, as of MySQL 3.23, you can retrieve rows in random order like this:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

`ORDER BY RAND()` combined with `LIMIT` is useful for selecting a random sample from a set of rows:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d -> ORDER BY RAND() LIMIT 1000;
```

`RAND()` [822] is not meant to be a perfect random generator. It is a fast way to generate random numbers on demand that is portable between platforms for the same MySQL version.

- `ROUND(X)` [823], `ROUND(X,D)` [823]

Rounds the argument *X* to *D* decimal places. *D* defaults to 0 if not specified. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT ROUND(-1.23);
        -> -1
mysql> SELECT ROUND(-1.58);
        -> -2
mysql> SELECT ROUND(1.58);
        -> 2
mysql> SELECT ROUND(1.298, 1);
        -> 1.3
mysql> SELECT ROUND(1.298, 0);
        -> 1
mysql> SELECT ROUND(23.298, -1);
        -> 20
```

The return type is the same type as that of the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places).

The behavior of `ROUND()` [823] when the argument is halfway between two integers depends on the C library implementation. Different implementations round to the nearest even number, always up, always down, or always toward zero. If you need one kind of rounding, you should use a well-defined function such as `TRUNCATE()` [825] or `FLOOR()` [820] instead.

- `SIGN(X)` [824]

  Returns the sign of the argument as `-1`, `0`, or `1`, depending on whether *X* is negative, zero, or positive.

  ```
  mysql> SELECT SIGN(-32);
          -> -1
  mysql> SELECT SIGN(0);
          -> 0
  mysql> SELECT SIGN(234);
          -> 1
  ```

- `SIN(X)` [824]

  Returns the sine of *X*, where *X* is given in radians.

  ```
  mysql> SELECT SIN(PI());
          -> 1.2246063538224e-16
  mysql> SELECT ROUND(SIN(PI()));
          -> 0
  ```

- `SQRT(X)` [824]

  Returns the square root of a nonnegative number *X*.

  ```
  mysql> SELECT SQRT(4);
          -> 2
  mysql> SELECT SQRT(20);
          -> 4.4721359549996
  mysql> SELECT SQRT(-16);
          -> NULL
  ```

- `TAN(X)` [824]

  Returns the tangent of *X*, where *X* is given in radians.

```
mysql> SELECT TAN(PI());
        -> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
        -> 1.5574077246549
```

- `TRUNCATE(X,D)` [825]

  Returns the number *X*, truncated to *D* decimal places. If *D* is `0`, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
        -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
        -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
        -> 1
mysql> SELECT TRUNCATE(-1.999,1);
        -> -1.9
mysql> SELECT TRUNCATE(122,-2);
       -> 100
mysql> SELECT TRUNCATE(10.28*100,0);
       -> 1027
```

  Starting from MySQL 3.23.51, all numbers are rounded toward zero.

11.7 Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 10.3, “Date and Time Types”, for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Table 11.13 Date/Time Functions

Name	Description
`ADDDATE()` [827]	Add time values (intervals) to a date value
`ADDTIME()` [828]	Add time
`CONVERT_TZ()` [828]	Convert from one timezone to another
`CURDATE()` [828]	Return the current date
`CURRENT_DATE()`, `CURRENT_DATE` [829]	Synonyms for CURDATE()
`CURRENT_TIME()`, `CURRENT_TIME` [829]	Synonyms for CURTIME()
`CURRENT_TIMESTAMP()`, `CURRENT_TIMESTAMP` [829]	Synonyms for NOW()
`CURTIME()` [829]	Return the current time
`DATE_ADD()` [829]	Add time values (intervals) to a date value
`DATE_FORMAT()` [832]	Format date as specified
`DATE_SUB()` [833]	Subtract a time value (interval) from a date
`DATE()` [829]	Extract the date part of a date or datetime expression
`DATEDIFF()` [829]	Subtract two dates

Name	Description
DAY() [833]	Synonym for DAYOFMONTH()
DAYNAME() [833]	Return the name of the weekday
DAYOFMONTH() [833]	Return the day of the month (0-31)
DAYOFWEEK() [834]	Return the weekday index of the argument
DAYOFYEAR() [834]	Return the day of the year (1-366)
EXTRACT() [834]	Extract part of a date
FROM_DAYS() [834]	Convert a day number to a date
FROM_UNIXTIME() [834]	Format UNIX timestamp as a date
GET_FORMAT() [835]	Return a date format string
HOUR() [836]	Extract the hour
LAST_DAY [836]	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME [836]	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP() [836]	Synonym for NOW()
MAKEDATE() [836]	Create a date from the year and day of year
MAKETIME() [836]	Create time from hour, minute, second
MICROSECOND() [837]	Return the microseconds from argument
MINUTE() [837]	Return the minute from the argument
MONTH() [837]	Return the month from the date passed
MONTHNAME() [837]	Return the name of the month
NOW() [837]	Return the current date and time
PERIOD_ADD() [837]	Add a period to a year-month
PERIOD_DIFF() [838]	Return the number of months between periods
QUARTER() [838]	Return the quarter from a date argument
SEC_TO_TIME() [838]	Converts seconds to 'HH:MM:SS' format
SECOND() [838]	Return the second (0-59)
STR_TO_DATE() [838]	Convert a string to a date
SUBDATE() [839]	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME() [840]	Subtract times
SYSDATE() [840]	Return the time at which the function executes
TIME_FORMAT() [840]	Format as time
TIME_TO_SEC() [841]	Return the argument converted to seconds
TIME() [840]	Extract the time portion of the expression passed
TIMEDIFF() [840]	Subtract time
TIMESTAMP() [840]	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TO_DAYS() [841]	Return the date argument converted to days
UNIX_TIMESTAMP() [841]	Return a UNIX timestamp

Name	Description
UTC_DATE() [842]	Return the current UTC date
UTC_TIME() [842]	Return the current UTC time
UTC_TIMESTAMP() [843]	Return the current UTC date and time
WEEK() [843]	Return the week number
WEEKDAY() [844]	Return the weekday index
WEEKOFYEAR() [844]	Return the calendar week of the date (0-53)
YEAR() [844]	Return the year
YEARWEEK() [844]	Return the year and week

Here is an example that uses date functions. The following query selects all rows with a *date_col* value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
    -> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

The query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as NOW() [837] within a single query always produce the same result. This principle also applies to CURDATE() [828], CURTIME() [829], UTC_DATE() [842], UTC_TIME() [842], UTC_TIMESTAMP() [843], and to any of their synonyms.

Beginning with MySQL 4.1.3, the CURRENT_TIMESTAMP() [829], CURRENT_TIME() [829], CURRENT_DATE() [829], and FROM_UNIXTIME() [834] functions return values in the connection's current time zone, which is available as the value of the time_zone [431] system variable. In addition, UNIX_TIMESTAMP() [841] assumes that its argument is a datetime value in the current time zone. See Section 9.7, “MySQL Server Time Zone Support”.

Some date functions can be used with “zero” dates or incomplete dates such as '2001-11-00', whereas others cannot. Functions that extract parts of dates typically work with incomplete dates and thus can return 0 when you might otherwise expect a nonzero value. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
       -> 0, 0
```

Other functions expect complete dates and return NULL for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
       -> NULL
mysql> SELECT DAYNAME('2006-05-00');
       -> NULL
```

- ADDDATE(*date*,INTERVAL *expr unit*) [827], ADDDATE(*expr*,*days*) [827]

  When invoked with the INTERVAL form of the second argument, ADDDATE() [827] is a synonym for DATE_ADD() [829]. The related function SUBDATE() [839] is a synonym for DATE_SUB() [833]. For information on the INTERVAL *unit* argument, see the discussion for DATE_ADD() [829].

```
mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
        -> '2008-02-02'
mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
        -> '2008-02-02'
```

As of MySQL 4.1.1, the second syntax is permitted. When invoked with the *days* form of the second argument, MySQL treats it as an integer number of days to be added to *expr*.

```
mysql> SELECT ADDDATE('2008-01-02', 31);
        -> '2008-02-02'
```

- `ADDTIME(expr1,expr2)` [828]

  `ADDTIME()` [828] adds *expr2* to *expr1* and returns the result. *expr1* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
        -> '2008-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
        -> '03:00:01.999997'
```

  `ADDTIME()` [828] was added in MySQL 4.1.1.

- `CONVERT_TZ(dt,from_tz,to_tz)` [828]

  `CONVERT_TZ()` [828] converts a datetime value *dt* from the time zone given by *from_tz* to the time zone given by *to_tz* and returns the resulting value. Time zones are specified as described in Section 9.7, “MySQL Server Time Zone Support”. This function returns `NULL` if the arguments are invalid.

  If the value falls out of the supported range of the `TIMESTAMP` type when converted from *from_tz* to UTC, no conversion occurs. The `TIMESTAMP` range is described in Section 10.1.2, “Date and Time Type Overview”.

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
        -> '2004-01-01 13:00:00'
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
        -> '2004-01-01 22:00:00'
```

Note

To use named time zones such as `'MET'` or `'Europe/Moscow'`, the time zone tables must be properly set up. See Section 9.7, “MySQL Server Time Zone Support”, for instructions.

  `CONVERT_TZ()` [828] was added in MySQL 4.1.3.

  If you intend to use `CONVERT_TZ()` [828] while other tables are locked with `LOCK TABLES`, you must also lock the `mysql.time_zone_name` table.

- `CURDATE()` [828]

  Returns the current date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
        -> '2008-06-13'
```

```
mysql> SELECT CURDATE() + 0;
        -> 20080613
```

- `CURRENT_DATE` [829], `CURRENT_DATE()` [829]

  `CURRENT_DATE` [829] and `CURRENT_DATE()` [829] are synonyms for `CURDATE()` [828].

- `CURTIME()` [829]

  Returns the current time as a value in `'HH:MM:SS'` or `HHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. (There is no `.uuuuuu` part before MySQL 4.1.13.) The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026.000000
```

- `CURRENT_TIME` [829], `CURRENT_TIME()` [829]

  `CURRENT_TIME` [829] and `CURRENT_TIME()` [829] are synonyms for `CURTIME()` [829].

- `CURRENT_TIMESTAMP` [829], `CURRENT_TIMESTAMP()` [829]

  `CURRENT_TIMESTAMP` [829] and `CURRENT_TIMESTAMP()` [829] are synonyms for `NOW()` [837].

- `DATE(`*`expr`*`)` [829]

  Extracts the date part of the date or datetime expression *`expr`*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

  `DATE()` [829] is available as of MySQL 4.1.1.

- `DATEDIFF(`*`expr1`*`,`*`expr2`*`)` [829]

  `DATEDIFF()` [829] returns *`expr1`* – *`expr2`* expressed as a value in days from one date to the other. *`expr1`* and *`expr2`* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
        -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
        -> -31
```

  `DATEDIFF()` [829] was added in MySQL 4.1.1.

- `DATE_ADD(`*`date`*`,INTERVAL `*`expr unit`*`)` [829], `DATE_SUB(`*`date`*`,INTERVAL `*`expr unit`*`)` [833]

  These functions perform date arithmetic. The *`date`* argument specifies the starting date or datetime value. *`expr`* is an expression specifying the interval value to be added or subtracted from the starting date. *`expr`* is a string; it may start with a "`-`" for negative intervals. *`unit`* is a keyword indicating the units in which the expression should be interpreted.

  The `INTERVAL` keyword and the *`unit`* specifier are not case sensitive.

The following table shows the expected form of the *expr* argument for each *unit* value.

unit Value	Expected *expr* Format	Version
MICROSECOND	MICROSECONDS	4.1.1
SECOND	SECONDS	Pre-4.1
MINUTE	MINUTES	Pre-4.1
HOUR	HOURS	Pre-4.1
DAY	DAYS	Pre-4.1
WEEK	WEEKS	5.0.0
MONTH	MONTHS	Pre-4.1
QUARTER	QUARTERS	5.0.0
YEAR	YEARS	Pre-4.1
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'	4.1.1
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'	4.1.1
MINUTE_SECOND	'MINUTES:SECONDS'	4.1.1
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'	4.1.1
HOUR_SECOND	'HOURS:MINUTES:SECONDS'	4.1.1
HOUR_MINUTE	'HOURS:MINUTES'	Pre-4.1
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'	4.1.1
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'	Pre-4.1
DAY_MINUTE	'DAYS HOURS:MINUTES'	Pre-4.1
DAY_HOUR	'DAYS HOURS'	Pre-4.1
YEAR_MONTH	'YEARS-MONTHS'	Pre-4.1

The *type* values DAY_MICROSECOND, HOUR_MICROSECOND, MINUTE_MICROSECOND, SECOND_MICROSECOND, and MICROSECOND are permitted as of MySQL 4.1.1.

MySQL permits any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a DATE value and your calculations involve only YEAR, MONTH, and DAY parts (that is, no time parts), the result is a DATE value. Otherwise, the result is a DATETIME value.

As of MySQL 3.23, date arithmetic also can be performed using INTERVAL together with the + [815] or - [815] operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

INTERVAL *expr unit* is permitted on either side of the + [815] operator if the expression on the other side is a date or datetime value. For the - [815] operator, INTERVAL *expr unit* is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '2009-01-01 00:00:00'
```

```
mysql> SELECT INTERVAL 1 DAY + '2008-12-31';
        -> '2009-01-01'
mysql> SELECT '2005-01-01' - INTERVAL 1 SECOND;
        -> '2004-12-31 23:59:59'
mysql> SELECT DATE_ADD('2000-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2001-01-01 00:00:00'
mysql> SELECT DATE_ADD('2010-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2011-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2005-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2004-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *`unit`* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a *`unit`* of `DAY_SECOND`, the value of *`expr`* is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10' DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as a time of day.

Because *`expr`* is treated as a string, be careful if you specify a nonstring value with `INTERVAL`. For example, with an interval specifier of `HOUR_MINUTE`, `6/4` evaluates to `1.50` and is treated as 1 hour, 50 minutes:

```
mysql> SELECT 6/4;
        -> 1.50
mysql> SELECT DATE_ADD('2009-01-01', INTERVAL 6/4 HOUR_MINUTE);
        -> '2009-01-04 12:20:00'
```

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 DAY);
        -> '2013-01-02'
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 HOUR);
        -> '2013-01-01 01:00:00'
```

If you add `MONTH`, `YEAR_MONTH`, or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('2009-01-30', INTERVAL 1 MONTH);
        -> '2009-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as `'2006-07-00'` or badly malformed dates:

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);
        -> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
        -> NULL
```

- `DATE_FORMAT(date,format)` [832]

  Formats the *`date`* value according to the *`format`* string.

  The following specifiers may be used in the *`format`* string. As of MySQL 3.23, the "`%`" character is required before format specifier characters. In earlier versions of MySQL, "`%`" was optional.

Specifier	Description
`%a`	Abbreviated weekday name (`Sun`..`Sat`)
`%b`	Abbreviated month name (`Jan`..`Dec`)
`%c`	Month, numeric (`0`..`12`)
`%D`	Day of the month with English suffix (`0th`, `1st`, `2nd`, `3rd`, …)
`%d`	Day of the month, numeric (`00`..`31`)
`%e`	Day of the month, numeric (`0`..`31`)
`%f`	Microseconds (`000000`..`999999`)
`%H`	Hour (`00`..`23`)
`%h`	Hour (`01`..`12`)
`%I`	Hour (`01`..`12`)
`%i`	Minutes, numeric (`00`..`59`)
`%j`	Day of year (`001`..`366`)
`%k`	Hour (`0`..`23`)
`%l`	Hour (`1`..`12`)
`%M`	Month name (`January`..`December`)
`%m`	Month, numeric (`00`..`12`)
`%p`	`AM` or `PM`
`%r`	Time, 12-hour (`hh:mm:ss` followed by `AM` or `PM`)
`%S`	Seconds (`00`..`59`)
`%s`	Seconds (`00`..`59`)
`%T`	Time, 24-hour (`hh:mm:ss`)
`%U`	Week (`00`..`53`), where Sunday is the first day of the week
`%u`	Week (`00`..`53`), where Monday is the first day of the week
`%V`	Week (`01`..`53`), where Sunday is the first day of the week; used with `%X`
`%v`	Week (`01`..`53`), where Monday is the first day of the week; used with `%x`
`%W`	Weekday name (`Sunday`..`Saturday`)
`%w`	Day of the week (`0`=Sunday..`6`=Saturday)
`%X`	Year for the week where Sunday is the first day of the week, numeric, four digits; used with `%V`

Specifier	Description
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal "%" character
%*x*	*x*, for any "*x*" not listed above

The %v, %V, %x, and %X format specifiers are available as of MySQL 3.23.8. %f is available as of MySQL 4.1.1.

Ranges for the month and day specifiers begin with zero due to the fact that MySQL permits the storing of incomplete dates such as '2014-00-00' (as of MySQL 3.23).

As of MySQL 4.1.21, the language used for day and month names and abbreviations is controlled by the value of the lc_time_names [417] system variable (Section 9.8, "MySQL Server Locale Support").

As of MySQL 4.1.23, DATE_FORMAT() [832] returns a string with a character set and collation given by character_set_connection [408] and collation_connection [409] so that it can return month and weekday names containing non-ASCII characters. Before 4.1.23, the return value is a binary string.

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
        -> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
        -> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
    ->                 '%D %y %a %d %m %b %j');
        -> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
    ->                 '%H %k %I %r %T %S %w');
        -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
        -> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
        -> '00'
```

- DATE_SUB(*date*,INTERVAL *expr unit*) [833]

  See the description for DATE_ADD() [829].

- DAY(*date*) [833]

  DAY() [833] is a synonym for DAYOFMONTH() [833]. It is available as of MySQL 4.1.1.

- DAYNAME(*date*) [833]

  Returns the name of the weekday for *date*. As of MySQL 4.1.21, the language used for the name is controlled by the value of the lc_time_names [417] system variable (Section 9.8, "MySQL Server Locale Support").

```
mysql> SELECT DAYNAME('2007-02-03');
        -> 'Saturday'
```

- DAYOFMONTH(*date*) [833]

Returns the day of the month for *date*, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero day part.

```
mysql> SELECT DAYOFMONTH('2007-02-03');
        -> 3
```

- DAYOFWEEK(*date*) [834]

  Returns the weekday index for *date* (1 = Sunday, 2 = Monday, …, 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('2007-02-03');
        -> 7
```

- DAYOFYEAR(*date*) [834]

  Returns the day of the year for *date*, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('2007-02-03');
        -> 34
```

- EXTRACT(*unit* FROM *date*) [834]

  The EXTRACT() [834] function uses the same kinds of unit specifiers as DATE_ADD() [829] or DATE_SUB() [833], but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '2009-07-02');
       -> 2009
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
       -> 200907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03');
       -> 20102
mysql> SELECT EXTRACT(MICROSECOND
    ->                FROM '2003-01-02 10:30:00.000123');
        -> 123
```

  EXTRACT() [834] was added in MySQL 3.23.0.

- FROM_DAYS(*N*) [834]

  Given a day number *N*, returns a DATE value.

```
mysql> SELECT FROM_DAYS(730669);
        -> '2007-07-03'
```

  Use FROM_DAYS() [834] with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See Section 11.8, “What Calendar Is Used By MySQL?”.

- FROM_UNIXTIME(*unix_timestamp*) [834], FROM_UNIXTIME(*unix_timestamp*,*format*) [834]

  Returns a representation of the *unix_timestamp* argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. (There is no .uuuuuu part before MySQL 4.1.13.) The value is expressed in the current time zone. *unix_timestamp* is an internal timestamp value such as is produced by the UNIX_TIMESTAMP() [841] function.

If *format* is given, the result is formatted according to the *format* string, which is used the same way as listed in the entry for the DATE_FORMAT() [832] function.

```
mysql> SELECT FROM_UNIXTIME(1196440219);
        -> '2007-11-30 10:30:19'
mysql> SELECT FROM_UNIXTIME(1196440219) + 0;
        -> 20071130103019.000000
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
    ->                      '%Y %D %M %h:%i:%s %x');
        -> '2007 30th November 10:30:59 2007'
```

Note: If you use UNIX_TIMESTAMP() [841] and FROM_UNIXTIME() [834] to convert between TIMESTAMP values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the UNIX_TIMESTAMP() [841] function.

- GET_FORMAT({DATE|TIME|DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'}) [835]

  Returns a format string. This function is useful in combination with the DATE_FORMAT() [832] and the STR_TO_DATE() [838] functions.

  The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the DATE_FORMAT() [832] function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
GET_FORMAT(DATE,'USA') [835]	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS') [835]	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO') [835]	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR') [835]	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL') [835]	'%Y%m%d'
GET_FORMAT(DATETIME,'USA') [835]	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS') [835]	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO') [835]	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR') [835]	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'INTERNAL') [835]	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA') [835]	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS') [835]	'%H:%i:%s'
GET_FORMAT(TIME,'ISO') [835]	'%H:%i:%s'
GET_FORMAT(TIME,'EUR') [835]	'%H.%i.%s'
GET_FORMAT(TIME,'INTERNAL') [835]	'%H%i%s'

As of MySQL 4.1.4, TIMESTAMP can also be used as the first argument to GET_FORMAT() [835], in which case the function returns the same values as for DATETIME.

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
        -> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
        -> '2003-10-31'
```

`GET_FORMAT()` [835] is available as of MySQL 4.1.1.

- `HOUR(time)` [836]

  Returns the hour for *`time`*. The range of the return value is `0` to `23` for time-of-day values. However, the range of `TIME` values actually is much larger, so `HOUR` can return values greater than `23`.

```
mysql> SELECT HOUR('10:05:03');
        -> 10
mysql> SELECT HOUR('272:59:59');
        -> 272
```

- `LAST_DAY(date)` [836]

  Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns `NULL` if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
        -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
        -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
        -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
        -> NULL
```

  `LAST_DAY()` [836] is available as of MySQL 4.1.1.

- `LOCALTIME` [836], `LOCALTIME()` [836]

  `LOCALTIME` [836] and `LOCALTIME()` [836] are synonyms for `NOW()` [837].

- `LOCALTIMESTAMP` [836], `LOCALTIMESTAMP()` [836]

  `LOCALTIMESTAMP` [836] and `LOCALTIMESTAMP()` [836] are synonyms for `NOW()` [837].

  They were added in MySQL 4.0.6.

- `MAKEDATE(year,dayofyear)` [836]

  Returns a date, given year and day-of-year values. *`dayofyear`* must be greater than 0 or the result is `NULL`.

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
        -> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
        -> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
        -> NULL
```

  `MAKEDATE()` [836] is available as of MySQL 4.1.1.

- `MAKETIME(hour,minute,second)` [836]

  Returns a time value calculated from the *`hour`*, *`minute`*, and *`second`* arguments.

```
mysql> SELECT MAKETIME(12,15,30);
        -> '12:15:30'
```

`MAKETIME()` [836] is available as of MySQL 4.1.1.

- `MICROSECOND(expr)` [837]

  Returns the microseconds from the time or datetime expression *`expr`* as a number in the range from `0` to `999999`.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('2009-12-31 23:59:59.000010');
        -> 10
```

  `MICROSECOND()` [837] is available as of MySQL 4.1.1.

- `MINUTE(time)` [837]

  Returns the minute for *`time`*, in the range `0` to `59`.

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
        -> 5
```

- `MONTH(date)` [837]

  Returns the month for *`date`*, in the range `1` to `12` for January to December, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` that have a zero month part.

```
mysql> SELECT MONTH('2008-02-03');
        -> 2
```

- `MONTHNAME(date)` [837]

  Returns the full name of the month for *`date`*. As of MySQL 4.1.21, the language used for the name is controlled by the value of the `lc_time_names` [417] system variable (Section 9.8, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('2008-02-03');
        -> 'February'
```

- `NOW()` [837]

  Returns the current date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. (There is no `.uuuuuu` part before MySQL 4.1.13.) The value is expressed in the current time zone.

```
mysql> SELECT NOW();
        -> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 20071215235026.000000
```

- `PERIOD_ADD(P,N)` [837]

  Adds *`N`* months to period *`P`* (in the format `YYMM` or `YYYYMM`). Returns a value in the format `YYYYMM`. Note that the period argument *`P`* is *not* a date value.

```
```

```
mysql> SELECT PERIOD_ADD(200801,2);
        -> 200803
```

- `PERIOD_DIFF(P1,P2)` [838]

  Returns the number of months between periods *`P1`* and *`P2`*. *`P1`* and *`P2`* should be in the format `YYMM` or `YYYYMM`. Note that the period arguments *`P1`* and *`P2`* are *not* date values.

```
mysql> SELECT PERIOD_DIFF(200802,200703);
        -> 11
```

- `QUARTER(date)` [838]

  Returns the quarter of the year for *`date`*, in the range `1` to `4`.

```
mysql> SELECT QUARTER('2008-04-01');
        -> 2
```

- `SECOND(time)` [838]

  Returns the second for *`time`*, in the range `0` to `59`.

```
mysql> SELECT SECOND('10:05:03');
        -> 3
```

- `SEC_TO_TIME(seconds)` [838]

  Returns the *`seconds`* argument, converted to hours, minutes, and seconds, as a `TIME` value. The range of the result is constrained to that of the `TIME` data type. A warning occurs if the argument corresponds to a value outside that range.

```
mysql> SELECT SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
        -> 3938
```

Note

You cannot use format `"%X%V"` to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
        -> '2004-10-18'
```

- `STR_TO_DATE(str,format)` [838]

  This is the inverse of the `DATE_FORMAT()` [832] function. It takes a string *`str`* and a format string *`format`*. `STR_TO_DATE()` [838] returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts. If the date, time, or datetime value extracted from *`str`* is illegal, `STR_TO_DATE()` [838] returns `NULL`.

  The server scans *`str`* attempting to match *`format`* to it. The format string can contain literal characters and format specifiers beginning with `%`. Literal characters in *`format`* must match literally in *`str`*. Format

specifiers in *format* must match a date or time part in *str*. For the specifiers that can be used in *format*, see the DATE_FORMAT() [832] function description.

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
        -> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
        -> '2013-05-01'
```

Scanning starts at the beginning of *str* and fails if *format* is found not to match. Extra characters at the end of *str* are ignored.

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
        -> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
        -> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
        -> '09:30:17'
```

Unspecified date or time parts have a value of 0, so incompletely specified values in *str* produce a result with some or all parts set to 0:

```
mysql> SELECT STR_TO_DATE('abc','abc');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
        -> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
        -> '00:00:09'
```

Range checking on the parts of date values is as described in Section 10.3.1, “The DATE, DATETIME, and TIMESTAMP Types”. This means, for example, that a date with a day part larger than the number of days in a month is permissible as long as the day part is in the range from 1 to 31. Also, “zero” dates or dates with part values of 0 are permitted.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
        -> '2004-04-31'
```

STR_TO_DATE() [838] is available as of MySQL 4.1.1.

- SUBDATE(*date*,INTERVAL *expr unit*) [839], SUBDATE(*expr*,*days*) [839]

  When invoked with the INTERVAL form of the second argument, SUBDATE() [839] is a synonym for DATE_SUB() [833]. For information on the INTERVAL *unit* argument, see the discussion for DATE_ADD() [829].

  ```
  mysql> SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
          -> '2007-12-02'
  mysql> SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
          -> '2007-12-02'
  ```

  As of MySQL 4.1.1, the second syntax is permitted, where *expr* is a date or datetime expression and *days* is the number of days to be subtracted from *expr*.

  ```
  mysql> SELECT SUBDATE('2008-01-02 12:00:00', 31);
          -> '2007-12-02 12:00:00'
  ```

- `SUBTIME(expr1,expr2)` [840]

  `SUBTIME()` [840] returns *`expr1`* – *`expr2`* expressed as a value in the same format as *`expr1`*. *`expr1`* is a time or datetime expression, and *`expr2`* is a time expression.

```
mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
        -> '2007-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

  `SUBTIME()` [840] was added in MySQL 4.1.1.

- `SYSDATE()` [840]

  Returns the current date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. (There is no `.uuuuuu` part before MySQL 4.1.13.)

- `TIME(expr)` [840]

  Extracts the time part of the time or datetime expression *`expr`* and returns it as a string.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
        -> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
        -> '01:02:03.000123'
```

  `TIME()` [840] is available as of MySQL 4.1.1.

- `TIMEDIFF(expr1,expr2)` [840]

  `TIMEDIFF()` [840] returns *`expr1`* – *`expr2`* expressed as a time value. *`expr1`* and *`expr2`* are time or date-and-time expressions, but both must be of the same type.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
    ->                 '2000:01:01 00:00:00.000001');
        -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',
    ->                 '2008-12-30 01:01:01.000002');
        -> '46:58:57.999999'
```

  `TIMEDIFF()` [840] was added in MySQL 4.1.1.

- `TIMESTAMP(expr)` [840], `TIMESTAMP(expr1,expr2)` [840]

  With a single argument, this function returns the date or datetime expression *`expr`* as a datetime value. With two arguments, it adds the time expression *`expr2`* to the date or datetime expression *`expr1`* and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
        -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
        -> '2004-01-01 00:00:00'
```

  `TIMESTAMP()` [840] is available as of MySQL 4.1.1.

- `TIME_FORMAT(time,format)` [840]

This is used like the DATE_FORMAT() [832] function, but the *format* string may contain format specifiers only for hours, minutes, seconds, and microseconds. Other specifiers produce a NULL value or 0.

If the *time* value contains an hour part that is greater than 23, the %H and %k hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
        -> '100 100 04 04 4'
```

- TIME_TO_SEC(*time*) [841]

  Returns the *time* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
        -> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
        -> 2378
```

- TO_DAYS(*date*) [841]

  Given a date *date*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
        -> 728779
mysql> SELECT TO_DAYS('2007-10-07');
        -> 733321
```

  TO_DAYS() [841] is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 11.8, “What Calendar Is Used By MySQL?”, for details.

  Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in Section 10.3, “Date and Time Types”. For example, '2008-10-07' and '08-10-07' are seen as identical dates:

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
        -> 733687, 733687
```

  In MySQL, the zero date is defined as '0000-00-00', even though this date is itself considered invalid. This means that, for '0000-00-00' and '0000-01-01', TO_DAYS() [841] returns the values shown here:

```
mysql> SELECT TO_DAYS('0000-00-00');
        -> NULL

mysql> SELECT TO_DAYS('0000-01-01');
        -> 1
```

- UNIX_TIMESTAMP() [841], UNIX_TIMESTAMP(*date*) [841]

  If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If UNIX_TIMESTAMP() [841] is called with a *date* argument, it returns

the value of the argument as seconds since `'1970-01-01 00:00:00'` UTC. *`date`* may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format `YYMMDD` or `YYYYMMDD`. The server interprets *`date`* as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in Section 9.7, “MySQL Server Time Zone Support”.

```
mysql> SELECT UNIX_TIMESTAMP();
        -> 1196440210
mysql> SELECT UNIX_TIMESTAMP('2007-11-30 10:30:19');
        -> 1196440219
```

When `UNIX_TIMESTAMP()` [841] is used on a `TIMESTAMP` column, the function returns the internal timestamp value directly, with no implicit “string-to-Unix-timestamp” conversion. If you pass an out-of-range date to `UNIX_TIMESTAMP()` [841], it returns `0`, but please note that only basic range checking is performed (year from `1970` to `2038`, month from `01` to `12`, day from `01` from `31`).

Note: If you use `UNIX_TIMESTAMP()` [841] and `FROM_UNIXTIME()` [834] to convert between `TIMESTAMP` values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes, it is possible for two `UNIX_TIMESTAMP()` [841] to map two `TIMESTAMP` values to the same Unix timestamp value. `FROM_UNIXTIME()` [834] will map that value back to only one of the original `TIMESTAMP` values. Here is an example, using `TIMESTAMP` values in the `CET` time zone:

```
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT FROM_UNIXTIME(1111885200);
+---------------------------+
| FROM_UNIXTIME(1111885200) |
+---------------------------+
| 2005-03-27 03:00:00       |
+---------------------------+
```

If you want to subtract `UNIX_TIMESTAMP()` [841] columns, you might want to cast the result to signed integers. See Section 11.10, “Cast Functions and Operators”.

- `UTC_DATE` [842], `UTC_DATE()` [842]

  Returns the current UTC date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

  ```
  mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
          -> '2003-08-14', 20030814
  ```

  `UTC_DATE()` [842] is available as of MySQL 4.1.1.

- `UTC_TIME` [842], `UTC_TIME()` [842]

Returns the current UTC time as a value in `'HH:MM:SS'` or `HHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. (There is no `.uuuuuu` part before MySQL 4.1.13.)

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
        -> '18:07:53', 180753.000000
```

`UTC_TIME()` [842] is available as of MySQL 4.1.1.

- `UTC_TIMESTAMP` [843], `UTC_TIMESTAMP()` [843]

  Returns the current UTC date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. (There is no `.uuuuuu` part before MySQL 4.1.13.)

  ```
  mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
          -> '2003-08-14 18:08:04', 20030814180804.000000
  ```

  `UTC_TIMESTAMP()` [843] is available as of MySQL 4.1.1.

- `WEEK(`*`date`*`[,`*`mode`*`])` [843]

  This function returns the week number for *`date`*. The two-argument form of `WEEK()` [843] enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from `0` to `53` or from `1` to `53`. If the *`mode`* argument is omitted, the value of the `default_week_format` [410] system variable is used (or `0` before MySQL 4.0.14). See Section 5.1.3, “Server System Variables”.

  The following table describes how the *`mode`* argument works.

Mode	First day of week	Range	Week 1 is the first week …
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with more than 3 days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with more than 3 days this year
4	Sunday	0-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with more than 3 days this year
7	Monday	1-53	with a Monday in this year

  A *`mode`* value of `3` can be used as of MySQL 4.0.5. Values of `4` and above can be used as of MySQL 4.0.17.

  ```
  mysql> SELECT WEEK('2008-02-20');
          -> 7
  mysql> SELECT WEEK('2008-02-20',0);
          -> 7
  mysql> SELECT WEEK('2008-02-20',1);
          -> 8
  mysql> SELECT WEEK('2008-12-31',1);
          -> 53
  ```

Note: In MySQL 4.0, `WEEK(date,0)` [843] was changed to match the calendar in the USA. Before that, `WEEK()` [843] was calculated incorrectly for dates in the USA. (In effect, `WEEK(date)` [843] and `WEEK(date,0)` [843] were incorrect for all cases.)

Note that if a date falls in the last week of the previous year, MySQL returns `0` if you do not use `2`, `3`, `6`, or `7` as the optional *`mode`* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
        -> 2000, 0
```

One might argue that MySQL should return `52` for the `WEEK()` [843] function, because the given date actually occurs in the 52nd week of 1999. We decided to return `0` instead because we want the function to return "the week number in the given year." This makes use of the `WEEK()` [843] function reliable when combined with other functions that extract a date part from a date.

If you would prefer the result to be evaluated with respect to the year that contains the first day of the week for the given date, use `0`, `2`, `5`, or `7` as the optional *`mode`* argument.

```
mysql> SELECT WEEK('2000-01-01',2);
        -> 52
```

Alternatively, use the `YEARWEEK()` [844] function:

```
mysql> SELECT YEARWEEK('2000-01-01');
        -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
        -> '52'
```

- `WEEKDAY(date)` [844]

  Returns the weekday index for *`date`* (`0` = Monday, `1` = Tuesday, … `6` = Sunday).

  ```
  mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
          -> 6
  mysql> SELECT WEEKDAY('2007-11-06');
          -> 1
  ```

- `WEEKOFYEAR(date)` [844]

  Returns the calendar week of the date as a number in the range from `1` to `53`. `WEEKOFYEAR()` [844] is a compatibility function that is equivalent to `WEEK(date,3)` [843].

  ```
  mysql> SELECT WEEKOFYEAR('2008-02-20');
          -> 8
  ```

  `WEEKOFYEAR()` [844] is available as of MySQL 4.1.1.

- `YEAR(date)` [844]

  Returns the year for *`date`*, in the range `1000` to `9999`, or `0` for the "zero" date.

  ```
  mysql> SELECT YEAR('1987-01-01');
          -> 1987
  ```

- `YEARWEEK(date)` [844], `YEARWEEK(date,mode)` [844]

Returns year and week for a date. The *mode* argument works exactly like the *mode* argument to WEEK() [843]. The year in the result may be different from the year in the date argument for the first and the last week of the year.

```
mysql> SELECT YEARWEEK('1987-01-01');
       -> 198653
```

Note that the week number is different from what the WEEK() [843] function would return (0) for optional arguments 0 or 1, as WEEK() [843] then returns the week in the context of the given year.

YEARWEEK() [844] was added in MySQL 3.23.8.

11.8 What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred.

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are nonexistent.

A calendar applied to dates when it was not actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL DATE or DATETIME values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

11.9 Full-Text Search Functions

MATCH (*col1,col2,...*) AGAINST (*expr* [*search_modifier*]) [845]

```
search_modifier: { IN BOOLEAN MODE | WITH QUERY EXPANSION }
```

As of MySQL 3.23.23, MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type FULLTEXT.
- Full-text indexes can be used only with MyISAM tables, and can be created only for CHAR, VARCHAR, or TEXT columns.

- A FULLTEXT index definition can be given in the CREATE TABLE statement when a table is created, or added later using ALTER TABLE or CREATE INDEX.
- For large data sets, it is much faster to load your data into a table that has no FULLTEXT index and then create the index after that, than to load data into a table that has an existing FULLTEXT index.

Full-text searching is performed using MATCH() ... AGAINST [845] syntax. MATCH() [845] takes a comma-separated list that names the columns to be searched. AGAINST takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a literal string, not a variable or a column name. There are three types of full-text searches:

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Common words such as “some” or “then” are stopwords and do not match if present in the search string. The IN BOOLEAN MODE modifier specifies a boolean search. For more information, see Section 11.9.2, “Boolean Full-Text Searches”.
- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators. The stopword list applies. In addition, words that are present in 50% or more of the rows are considered common and do not match. Full-text searches are natural language searches if no modifier is given.
- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The WITH QUERY EXPANSION modifier specifies a query expansion search. For more information, see Section 11.9.3, “Full-Text Searches with Query Expansion”.

Constraints on full-text searching are listed in Section 11.9.5, “Full-Text Restrictions”.

The myisam_ftdump utility can be used to dump the contents of a full-text index. This may be helpful for debugging full-text queries. See Section 4.6.1, “myisam_ftdump — Display Full-Text Index information”.

11.9.1 Natural Language Full-Text Searches

By default, the MATCH() [845] function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a FULLTEXT index. The search string is given as the argument to AGAINST(). For each row in the table, MATCH() [845] returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the MATCH() [845] list.

```
mysql> CREATE TABLE articles (
    ->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   title VARCHAR(200),
    ->   body TEXT,
    ->   FULLTEXT (title,body)
    -> ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
    -> ('MySQL Tutorial','DBMS stands for DataBase ...'),
    -> ('How To Use MySQL Well','After you went through a ...'),
    -> ('Optimizing MySQL','In this tutorial we will show ...'),
    -> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
    -> ('MySQL vs. YourSQL','In the following database comparison ...'),
    -> ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+---------------------------------------+
| id | title             | body                                  |
+----+-------------------+---------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...          |
+----+-------------------+---------------------------------------+
2 rows in set (0.00 sec)
```

By default, the search is performed in case-insensitive fashion. In MySQL 4.1 and up, you can make a full-text search by using a binary collation for the indexed columns. For example, a column that has a character set of `latin1` character set of can be assigned a collation of `latin1_bin` to make it case sensitive for full-text searches.

When `MATCH()` [845] is used in a `WHERE` clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are nonnegative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

To simply count matches, you could use a query like this:

```
mysql> SELECT COUNT(*) FROM articles
    -> WHERE MATCH (title,body)
    -> AGAINST ('database');
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

However, you might find it quicker to rewrite the query as follows:

```
mysql> SELECT
    -> COUNT(IF(MATCH (title,body) AGAINST ('database'), 1, NULL))
    -> AS count
    -> FROM articles;
+-------+
| count |
+-------+
|     2 |
+-------+
1 row in set (0.00 sec)
```

The first query sorts the results by relevance whereas the second does not. However, the second query performs a full table scan and the first does not. The first may be faster if the search matches few rows; otherwise, the second may be faster because it would read many rows anyway.

For natural-language full-text searches, it is a requirement that the columns named in the `MATCH()` [845] function be the same columns included in some `FULLTEXT` index in your table. For the preceding query, note that the columns named in the `MATCH()` [845] function (`title` and `body`) are the same as those named in the definition of the `article` table's `FULLTEXT` index. If you wanted to search the `title` or `body` separately, you would need to create separate `FULLTEXT` indexes for each column.

It is also possible to perform a boolean search or a search with query expansion. These search types are described in Section 11.9.2, “Boolean Full-Text Searches”, and Section 11.9.3, “Full-Text Searches with Query Expansion”.

A full-text search that uses an index can name columns only from a single table in the `MATCH()` [845] clause because an index cannot span multiple tables. A boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the `MATCH()` [845] function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the `SELECT` statement includes neither `WHERE` nor `ORDER BY` clauses:

```
mysql> SELECT id, MATCH (title,body) AGAINST ('Tutorial')
    -> FROM articles;
+----+----------------------------------------+
| id | MATCH (title,body) AGAINST ('Tutorial') |
+----+----------------------------------------+
|  1 |                        0.65545833110809 |
|  2 |                                       0 |
|  3 |                        0.66266459226608 |
|  4 |                                       0 |
|  5 |                                       0 |
|  6 |                                       0 |
+----+----------------------------------------+
6 rows in set (0.00 sec)
```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, specify `MATCH()` [845] twice: once in the `SELECT` list and once in the `WHERE` clause. This causes no additional overhead, because the MySQL optimizer notices that the two `MATCH()` [845] calls are identical and invokes the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root') AS score
    -> FROM articles WHERE MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root');
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

The MySQL `FULLTEXT` implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes (“`'`”), but not more than one in a row. This means that `aaa'bbb` is regarded as one word, but `aaa''bbb` is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the `FULLTEXT` parser; `'aaa'bbb'` would be parsed as `aaa'bbb`.

The `FULLTEXT` parser determines where words start and end by looking for certain delimiter characters; for example, “ ” (space), “`,`” (comma), and “`.`” (period). If words are not separated by delimiters (as in, for example, Chinese), the `FULLTEXT` parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a `FULLTEXT` index, you must preprocess them so that they are separated by some arbitrary delimiter such as “`"`”.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is four characters.

- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overwritten by a user-defined list.

The default stopword list is given in Section 11.9.4, “Full-Text Stopwords”. The default minimum word length and stopword list can be changed as described in Section 11.9.6, “Fine-Tuning MySQL Full-Text Search”.

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Consequently, a word that is present in many documents has a lower weight (and may even have a zero weight), because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row.

Such a technique works best with large collections (in fact, it was carefully tuned this way). For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results. For example, although the word “MySQL” is present in every row of the `articles` table shown earlier, a search for the word produces no results:

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)
```

The search result is empty because the word “MySQL” is present in at least 50% of the rows. As such, it is effectively treated as a stopword. For large data sets, this is the most desirable behavior: A natural language query should not return every second row from a 1GB table. For small data sets, it may be less desirable.

A word that matches half of the rows in a table is less likely to locate relevant documents. In fact, it most likely finds plenty of irrelevant documents. We all know this happens far too often when we are trying to find something on the Internet with a search engine. It is with this reasoning that rows containing the word are assigned a low semantic value for *the particular data set in which they occur*. A given word may reach the 50% threshold in one data set but not another.

The 50% threshold has a significant implication when you first try full-text searching to see how it works: If you create a table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results. Be sure to insert at least three rows, and preferably many more. Users who need to bypass the 50% limitation can use the boolean search mode; see Section 11.9.2, “Boolean Full-Text Searches”.

11.9.2 Boolean Full-Text Searches

As of version 4.0.1, MySQL can perform boolean full-text searches using the `IN BOOLEAN MODE` modifier. `IN BOOLEAN MODE` modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string. In the following query, the `+` and `-` operators indicate that a word is required to be present or absent, respectively, for a match to occur. Thus, the query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
    -> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+----+-----------------------+-------------------------------------+
| id | title                 | body                                |
+----+-----------------------+-------------------------------------+
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...        |
|  2 | How To Use MySQL Well | After you went through a ...        |
|  3 | Optimizing MySQL      | In this tutorial we will show ...   |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ... |
```

```
|  6 | MySQL Security         | When configured properly, MySQL ... |
+----+------------------------+--------------------------------------+
```

Note

In implementing this feature, MySQL uses what is sometimes referred to as *implied Boolean logic*, in which

- `+` stands for `AND`
- `-` stands for `NOT`
- [*no operator*] implies `OR`

Boolean full-text searches have these characteristics:

- They do not use the 50% threshold.
- They do not automatically sort rows in order of decreasing relevance. You can see this from the preceding query result: The row with the highest relevance is the one that contains “MySQL” twice, but it is listed last, not first.
- They can work even without a `FULLTEXT` index, although a search executed in this fashion would be quite slow.
- The minimum and maximum word length full-text parameters apply.
- The stopword list applies.

The boolean full-text search capability supports the following operators:

- `+`

  A leading plus sign indicates that this word *must* be present in each row that is returned.

- `-`

  A leading minus sign indicates that this word must *not* be present in any of the rows that are returned.

  Note: The `-` operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by `-` returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

  By default (when neither `+` nor `-` is specified) the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` [845] without the `IN BOOLEAN MODE` modifier.

- `> <`

  These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The `>` operator increases the contribution and the `<` operator decreases it. See the example following this list.

- `( )`

  Parentheses group words into subexpressions. Parenthesized groups can be nested.

- `~`

A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the `-` operator.

- `*`

  The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it should be *appended* to the word to be affected. Words match if they begin with the word preceding the `*` operator.

  If a word is specified with the truncation operator, it is not stripped from a boolean query, even if it is too short (as determined from the `ft_min_word_len` [412] setting) or a stopword. This occurs because the word is not seen as too short or a stopword, but as a prefix that must be present in the document in the form of a word that begins with the prefix. Suppose that `ft_min_word_len=4`. Then a search for `'+word +the*'` will likely return fewer rows than a search for `'+word +the'`:

  - The former query remains as is and requires both *`word`* and `the*` (a word starting with `the`) to be present in the document.
  - The latter query is transformed to `+word` (requiring only *`word`* to be present). `the` is both too short and a stopword, and either condition is enough to cause it to be ignored.

- `"`

  A phrase that is enclosed within double quote (“`"`”) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the `FULLTEXT` index for the words. The engine then performs a substring search for the phrase in the records that are found, so the match must include nonword characters in the phrase. For example, `"test phrase"` does *not* match `"test, phrase"`.

  If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

  Find rows that contain at least one of the two words.

- `'+apple +juice'`

  Find rows that contain both words.

- `'+apple macintosh'`

  Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- `'+apple -macintosh'`

  Find rows that contain the word “apple” but not “macintosh”.

- `'+apple ~macintosh'`

  Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for `'+apple -macintosh'`, for which the presence of “macintosh” causes the row not to be returned at all.

- `'+apple +(>turnover <strudel)'`

Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- `'apple*'`

  Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- `'"some words"'`

  Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the “`"`” characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotation marks that enclose the search string itself.

11.9.3 Full-Text Searches with Query Expansion

As of MySQL 4.1.1, full-text search supports query expansion (in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+---------------------------------------+
| id | title             | body                                  |
+----+-------------------+---------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...          |
+----+-------------------+---------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body)
    -> AGAINST ('database' WITH QUERY EXPANSION);
+----+-------------------+---------------------------------------+
| id | title             | body                                  |
+----+-------------------+---------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...          |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  3 | Optimizing MySQL  | In this tutorial we will show ...     |
+----+-------------------+---------------------------------------+
3 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

Note

Because blind query expansion tends to increase noise significantly by returning nonrelevant documents, it is meaningful to use only when a search phrase is rather short.

11.9.4 Full-Text Stopwords

The stopword list is loaded and searched for full-text queries using the server character set and collation (the values of the `character_set_server` and `collation_server` system variables). False hits or misses may occur for stopword lookups if the stopword file or columns used for full-text indexing or searches have a character set or collation different from `character_set_server` or `collation_server`.

Case sensitivity of stopword lookups depends on the server collation. For example, lookups are case insensitive if the collation is `latin1_swedish_ci`, whereas lookups are case sensitive if the collation is `latin1_general_cs` or `latin1_bin`.

The following table shows the default list of full-text stopwords. In a MySQL source distribution, you can find this list in the `myisam/ft_static.c` file.

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow
allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either

else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	known	knows	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours

ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who

who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
wouldn't	yes	yet	you	you'd
you'll	you're	you've	your	yours
yourself	yourselves	zero		

11.9.5 Full-Text Restrictions

- Full-text searches are supported for `MyISAM` tables only.
- As of MySQL 4.1.1, full-text searches can be used with most multi-byte character sets. The exception is that for Unicode, the `utf8` character set can be used, but not the `ucs2` character set. However, although `FULLTEXT` indexes on `ucs2` columns cannot be used, you can perform `IN BOOLEAN MODE` searches on a `ucs2` column that has no such index.
- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the `FULLTEXT` parser *cannot determine where words begin and end in these and other such languages*. The implications of this and some workarounds for the problem are described in Section 11.9, “Full-Text Search Functions”.
- As of MySQL 4.1, the use of multiple character sets within a single table is supported. However, all columns in a `FULLTEXT` index must use the same character set and collation.
- The `MATCH()` [845] column list must match exactly the column list in some `FULLTEXT` index definition for the table, unless this `MATCH()` [845] is `IN BOOLEAN MODE`. Boolean-mode searches can be done on nonindexed columns, although they are likely to be slow.
- The argument to `AGAINST()` must be a constant string.
- Index hints do not work for `FULLTEXT` searches.

11.9.6 Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See Section 2.9, “Installing MySQL from Source”.

Note that full-text search is carefully tuned for the most effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing*.

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the `FULLTEXT` indexes in your tables. Instructions for doing so are given later in this section.

- The minimum and maximum lengths of words to be indexed are defined by the `ft_min_word_len` [412] and `ft_max_word_len` [412] system variables (available as of MySQL 4.0.0). See Section 5.1.3, “Server System Variables”.) The default minimum value is four characters; the default maximum is version dependent. If you change either value, you must rebuild your `FULLTEXT` indexes. For example, if you want three-character words to be searchable, you can set the `ft_min_word_len` [412] variable by putting the following lines in an option file:

```
[mysqld]
ft_min_word_len=3
```

Then restart the server and rebuild your `FULLTEXT` indexes. Note particularly the remarks regarding `myisamchk` in the instructions following this list.

- To override the default stopword list, set the `ft_stopword_file` [412] system variable (available as of MySQL 4.0.10). See Section 5.1.3, “Server System Variables”.) The variable value should be the path name of the file containing the stopword list, or the empty string to disable stopword filtering. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your `FULLTEXT` indexes.

  The stopword list is free-form. That is, you may use any nonalphanumeric character such as newline, space, or comma to separate stopwords. Exceptions are the underscore character (“_”) and a single apostrophe (“`'`”) which are treated as part of a word. The character set of the stopword list is the server's default character set; see Section 9.1.3.1, “Server Character Set and Collation”.

- The 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in `myisam/ftdefs.h`:

  ```
  #define GWS_IN_USE GWS_PROB
  ```

  Change that line to this:

  ```
  #define GWS_IN_USE GWS_FREQ
  ```

  Then recompile MySQL. There is no need to rebuild the indexes in this case.

  Note

  By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the `MATCH()` [845] function. If you really need to search for such common words, it would be better to search using `IN BOOLEAN MODE` instead, which does not observe the 50% threshold.

- To change the operators used for boolean full-text searches, set the `ft_boolean_syntax` [411] system variable (available as of MySQL 4.0.1). The variable can be changed while the server is running, but you must have the `SUPER` [493] privilege to do so. No rebuilding of indexes is necessary in this case. See Section 5.1.3, “Server System Variables”, which describes the rules governing how to set this variable.

- If you want to change the set of characters that are considered word characters, you can do so in several ways, as described in the following list. After making the modification, you must rebuild the indexes for each table that contains any `FULLTEXT` indexes. Suppose that you want to treat the hyphen character ('-') as a word character. Use one of these methods:

  - Modify the MySQL source: In `myisam/ftdefs.h`, see the `true_word_char()` and `misc_word_char()` macros. Add `'-'` to one of those macros and recompile MySQL.

  - Modify a character set file: This requires no recompilation. The `true_word_char()` macro uses a “character type” table to distinguish letters and numbers from other characters. . You can edit the contents of the `<ctype><map>` array in one of the character set XML files to specify that `'-'` is a “letter.” Then use the given character set for your `FULLTEXT` indexes. For information about the `<ctype><map>` array format, see Section 9.4.1, “The Character Definition Arrays”.

- Add a new collation for the character set used by the indexed columns, and alter the columns to use that collation. For information about adding collations, see Section 9.5, “How to Add a New Collation to a Character Set”.

If you modify full-text variables that affect indexing (`ft_min_word_len` [412], `ft_max_word_len` [412], or `ft_stopword_file` [412]), or if you change the stopword file itself, you must rebuild your `FULLTEXT` indexes after making the changes and restarting the server. To rebuild the indexes in this case, it is sufficient to do a `QUICK` repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Alternatively, use `ALTER TABLE` with the `DROP INDEX` and `ADD INDEX` options to drop and re-create each `FULLTEXT` index. In some cases, this may be faster than a repair operation.

Each table that contains any `FULLTEXT` index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table will cause the server to see the table as corrupt and in need of repair.

With regard specifically to using the `IN BOOLEAN MODE` capability, if you upgrade from MySQL 3.23 to 4.0 or later, it is necessary to replace the index header as well. To do this, perform a `USE_FRM` repair operation:

```
mysql> REPAIR TABLE tbl_name USE_FRM;
```

This is necessary because boolean full-text searches require a flag in the index header that was not present in MySQL 3.23, and that is not added if you do only a `QUICK` repair. If you attempt a boolean full-text search without rebuilding the indexes this way, the search returns incorrect results.

Note that if you use `myisamchk` to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or stopword file values used by the server, specify the same `ft_min_word_len` [412], `ft_max_word_len` [412], and `ft_stopword_file` [412] values for `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` for index modification is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE` statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

11.10 Cast Functions and Operators

Table 11.14 Cast Functions

Name	Description
BINARY [859]	Cast a string to a binary string
CAST() [859]	Cast a value as a certain type
CONVERT() [859]	Cast a value as a certain type

- BINARY [859]

  The BINARY [859] operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column is not defined as BINARY or BLOB. BINARY [859] also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
mysql> SELECT 'a' = 'a ';
        -> 1
mysql> SELECT BINARY 'a' = 'a ';
        -> 0
```

  In a comparison, BINARY [859] affects the entire operation; it can be given before either operand with the same result.

  BINARY [859] was added in MySQL 3.23.0. As of MySQL 4.0.2, BINARY *str* [859] is a shorthand for CAST(*str* AS BINARY) [859].

  Note that in some contexts, if you cast an indexed column to BINARY, MySQL is not able to use the index efficiently.

- CAST(*expr* AS *type*) [859]

  The CAST() [859] function takes a value of one type and produce a value of another type, similar to CONVERT() [859]. See the description of CONVERT() [859] for more information.

- CONVERT(*expr*,*type*) [859], CONVERT(*expr* USING *transcoding_name*) [859]

  The CONVERT() [859] and CAST() [859] functions take a value of one type and produce a value of another type.

  The *type* can be one of the following values:

  - BINARY (and BINARY[*N*] as of MySQL 4.1.1)
  - CHAR (and CHAR[*N*] as of MySQL 4.1.1)
  - DATE
  - DATETIME
  - SIGNED [INTEGER]
  - TIME

- `UNSIGNED [INTEGER]`

`BINARY` produces a string with the `BINARY` data type. See Section 10.4.2, "The `BINARY` and `VARBINARY` Types" for a description of how this affects comparisons. If the optional length *`N`* is given, `BINARY(N)` causes the cast to use no more than *`N`* bytes of the argument. Similarly, `CHAR[N]` causes the cast to use no more than *`N`* characters of the argument.

`CAST()` [859] and `CONVERT()` [859] are available as of MySQL 4.0.2. The `CHAR` conversion type is available as of 4.0.6. The `USING` form of `CONVERT()` [859] is available as of 4.1.0.

`CAST()` [859] and `CONVERT(... USING ...)` [859] are standard SQL syntax. The non-`USING` form of `CONVERT()` [859] is ODBC syntax.

`CONVERT()` [859] with `USING` is used to convert data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string `'abc'` in the default character set to the corresponding string in the `utf8` character set:

```
SELECT CONVERT('abc' USING utf8);
```

If you want to compare a `BLOB` value or other binary string in case-insensitive fashion, you can do so as follows:

- Before MySQL 4.1.1, use the `UPPER()` [804] function to convert the binary string to uppercase before performing the comparison:

```
SELECT 'A' LIKE UPPER(blob_col) FROM tbl_name;
```

  If the comparison value is lowercase, convert the string value using `LOWER()` [798] instead.

- For MySQL 4.1.1 and up, binary strings have no character set, and thus no concept of lettercase. To perform a case-insensitive comparison, use the `CONVERT()` [859] function to convert the value to a nonbinary string. Comparisons of the result use the string collation. For example, if the character set of the result has a case-insensitive collation, a `LIKE` [804] operation is not case sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

  To use a different character set, substitute its name for `latin1` in the preceding statement. To specify a particular collation for the converted string, use a `COLLATE` clause following the `CONVERT()` [859] call, as described in Section 9.1.8.2, "`CONVERT()` and `CAST()`". For example, to use `latin1_german1_ci`:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) COLLATE latin1_german1_ci
  FROM tbl_name;
```

`CONVERT()` [859] can be used more generally for comparing strings that are represented in different character sets.

`LOWER()` [798] (and `UPPER()` [804]) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-------------+---------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
```

```
+-------------+----------------------------------+
| New York    | new york                         |
+-------------+----------------------------------+
```

The cast functions are useful when you want to create a column with a specific type in a `CREATE ... SELECT` statement:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

The functions also can be useful for sorting `ENUM` columns in lexical order. Normally, sorting of `ENUM` columns occurs using the internal numeric values. Casting the values to `CHAR` results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CAST(str AS BINARY)` [859] is the same thing as `BINARY str` [859]. `CAST(expr AS CHAR)` [859] treats the expression as a string with the default character set.

Note

In MySQL 4.0, a `CAST()` [859] to `DATE`, `DATETIME`, or `TIME` only marks the column to be a specific type but does not change the value of the column.

As of MySQL 4.1.0, the value is converted to the correct column type when it is sent to the user (this is a feature of how the new protocol in 4.1 sends date information to the client):

```
mysql> SELECT CAST(NOW() AS DATE);
       -> 2003-05-26
```

As of MySQL 4.1.1, `CAST()` [859] also changes the result if you use it as part of a more complex expression such as `CONCAT('Date: ',CAST(NOW() AS DATE))`.

You should not use `CAST()` [859] to extract data in different formats but instead use string functions like `LEFT()` [797] or `EXTRACT()` [834]. See Section 11.7, “Date and Time Functions”.

To cast a string to a numeric value in numeric context, you normally do not have to do anything other than to use the string value as though it were a number:

```
mysql> SELECT 1+'1';
       -> 2
```

If you use a string in an arithmetic operation, it is converted to a floating-point number during expression evaluation.

If you use a number in string context, the number automatically is converted to a string:

```
mysql> SELECT CONCAT('hello you ',2);
        -> 'hello you 2'
```

For information about implicit conversion of numbers to strings, see Section 11.2, “Type Conversion in Expression Evaluation”.

MySQL supports arithmetic with both signed and unsigned 64-bit values. If you are using numeric operators (such as `+` [815] or `-` [815]) and one of the operands is an unsigned integer, the result

is unsigned by default (see Section 11.6.1, “Arithmetic Operators”). You can override this by using the `SIGNED` or `UNSIGNED` cast operator to cast a value to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
        -> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
        -> -1
```

If either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, `DECIMAL` column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
        -> -1.0
```

The handing of unsigned values was changed in MySQL 4.0 to be able to support `BIGINT` values properly. If you have some code that you want to run in both MySQL 4.0 and 3.23, you probably cannot use the `CAST()` [859] function. You can use the following technique to get a signed result when subtracting two unsigned integer columns `ucol1` and `ucol2`:

```
mysql> SELECT (ucol1+0.0)-(ucol2+0.0) FROM ...;
```

The idea is that the columns are converted to floating-point values before the subtraction occurs.

If you have a problem with `UNSIGNED` columns in old MySQL applications when porting them to MySQL 4.0, you can use the `--sql-mode=NO_UNSIGNED_SUBTRACTION` [394] option when starting `mysqld`. However, as long as you use this option, you are not able to make efficient use of the `BIGINT UNSIGNED` data type.

11.11 Bit Functions

Table 11.15 Bitwise Functions

Name	Description
`BIT_COUNT()` [863]	Return the number of bits that are set
`&` [863]	Bitwise AND
`~` [863]	Invert bits
`|` [862]	Bitwise OR
`^` [863]	Bitwise XOR
`<<` [863]	Left shift
`>>` [863]	Right shift

MySQL uses `BIGINT` (64-bit) arithmetic for bit operations, so these operators have a maximum range of 64 bits.

- `|` [862]

  Bitwise OR:

  ```
  mysql> SELECT 29 | 15;
          -> 31
  ```

The result is an unsigned 64-bit integer.

- `&` [863]

  Bitwise AND:

```
mysql> SELECT 29 & 15;
        -> 13
```

  The result is an unsigned 64-bit integer.

- `^` [863]

  Bitwise XOR:

```
mysql> SELECT 1 ^ 1;
        -> 0
mysql> SELECT 1 ^ 0;
        -> 1
mysql> SELECT 11 ^ 3;
        -> 8
```

  The result is an unsigned 64-bit integer.

  Bitwise XOR was added in MySQL 4.0.2.

- `<<` [863]

  Shifts a longlong (`BIGINT`) number to the left.

```
mysql> SELECT 1 << 2;
        -> 4
```

  The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- `>>` [863]

  Shifts a longlong (`BIGINT`) number to the right.

```
mysql> SELECT 4 >> 2;
        -> 1
```

  The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- `~` [863]

  Invert all bits.

```
mysql> SELECT 5 & ~1;
        -> 4
```

  The result is an unsigned 64-bit integer.

- `BIT_COUNT(N)` [863]

Returns the number of bits that are set in the argument *N*.

```
mysql> SELECT BIT_COUNT(29);
        -> 4
```

11.12 Encryption and Compression Functions

Table 11.16 Encryption Functions

Name	Description
AES_DECRYPT() [865]	Decrypt using AES
AES_ENCRYPT() [865]	Encrypt using AES
COMPRESS() [866]	Return result as a binary string
DECODE() [866]	Decodes a string encrypted using ENCODE()
DES_DECRYPT() [866]	Decrypt a string
DES_ENCRYPT() [866]	Encrypt a string
ENCODE() [867]	Encode a string
ENCRYPT() [867]	Encrypt a string
MD5() [868]	Calculate MD5 checksum
OLD_PASSWORD() [868]	Return the value of the pre-4.1 implementation of PASSWORD
PASSWORD() [868]	Calculate and return a password string
SHA1(), SHA() [869]	Calculate an SHA-1 160-bit checksum
UNCOMPRESS() [869]	Uncompress a string compressed
UNCOMPRESSED_LENGTH() [869]	Return the length of a string before compression

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a BLOB binary string data type. This will avoid potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type (CHAR, VARCHAR, TEXT).

For functions such as MD5() or SHA1() that return a string of hex digits, the return value cannot be converted to uppercase or compared in case-insensitive fashion as is. You must convert the value to a nonbinary string. See the discussion of binary string conversion in Section 11.10, “Cast Functions and Operators”.

If an application stores values from a function such as MD5() [868] or SHA1() [869] that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using UNHEX() [803] and storing the result in a BINARY(*N*) column. Each pair of hex digits requires one byte in binary form, so the value of *N* depends on the length of the hex string. *N* is 16 for an MD5() [868] value and 20 for a SHA1() [869] value.

The size penalty for storing the hex string in a CHAR column is at least two times, up to six times if the value is stored in a column that uses the utf8 character set (where each character uses 3 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores MD5() [868] string values in a CHAR(32) column:

```
```

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

To convert hex strings to more compact form, modify the application to use `UNHEX()` [803] and `BINARY(16)` instead as follows:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Applications should be prepared to handle the very rare case that a hashing function produces the same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.

Note

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using one of the other encryption functions described in this section instead.

Caution

Passwords or other sensitive values supplied as arguments to encryption functions are sent in plaintext to the MySQL server unless an SSL connection is used. Also, such values will appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

- `AES_DECRYPT(crypt_str,key_str)` [865]

  This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of `AES_ENCRYPT()` [865].

- `AES_ENCRYPT(str,key_str)` [865]

  `AES_ENCRYPT()` [865] and `AES_DECRYPT()` [865] enable encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” Encoding with a 128-bit key length is used, but you can extend it up to 256 bits by modifying the source. We chose 128 bits because it is much faster and it is secure enough for most purposes.

  `AES_ENCRYPT()` [865] encrypts a string and returns a binary string. `AES_DECRYPT()` [865] decrypts the encrypted string and returns the original string. The input arguments may be any length. If either argument is `NULL`, the result of this function is also `NULL`.

  Because AES is a block-level algorithm, padding is used to encode uneven length strings and so the result string length may be calculated using this formula:

  ```
  16 * (trunc(string_length / 16) + 1)
  ```

  If `AES_DECRYPT()` [865] detects invalid data or incorrect padding, it returns `NULL`. However, it is possible for `AES_DECRYPT()` [865] to return a non-`NULL` value (possibly garbage) if the input data or the key is invalid.

  You can use the AES functions to store data in an encrypted form by modifying your queries:

  ```
  ```

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

`AES_ENCRYPT()` [865] and `AES_DECRYPT()` [865] were added in MySQL 4.0.2, and can be considered the most cryptographically secure encryption functions available in MySQL.

- `COMPRESS(string_to_compress)` [866]

  Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with `UNCOMPRESS()` [869].

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
        -> 21
mysql> SELECT LENGTH(COMPRESS(''));
        -> 0
mysql> SELECT LENGTH(COMPRESS('a'));
        -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
        -> 15
```

  The compressed string contents are stored the following way:

  - Empty strings are stored as empty strings.
  - Nonempty strings are stored as a four-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra "`.`" character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (However, use of nonbinary string data types such as `CHAR` or `VARCHAR` to store compressed strings is not recommended anyway because character set conversion may occur. Use a `VARBINARY` or `BLOB` binary string column instead.)

  `COMPRESS()` [866] was added in MySQL 4.1.1.

- `DECODE(crypt_str,pass_str)` [866]

  Decrypts the encrypted string *`crypt_str`* using *`pass_str`* as the password. *`crypt_str`* should be a string returned from `ENCODE()` [867].

- `DES_DECRYPT(crypt_str[,key_str])` [866]

  Decrypts a string encrypted with `DES_ENCRYPT()` [866]. If an error occurs, this function returns `NULL`.

  This function works only if MySQL has been configured with SSL support. See Section 5.6.6, "Using SSL for Secure Connections".

  If no *`key_str`* argument is given, `DES_DECRYPT()` [866] examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the `SUPER` [493] privilege. The key file can be specified with the `--des-key-file` [386] server option.

  If you pass this function a *`key_str`* argument, that string is used as the key for decrypting the message.

  If the *`crypt_str`* argument does not appear to be an encrypted string, MySQL returns the given *`crypt_str`*.

  `DES_DECRYPT()` [866] was added in MySQL 4.0.1.

- `DES_ENCRYPT(str[,{key_num|key_str}])` [866]

Encrypts the string with the given key using the Triple-DES algorithm.

This function works only if MySQL has been configured with SSL support. See Section 5.6.6, “Using SSL for Secure Connections”.

The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()` [866], if one was given. With no argument, the first key from the DES key file is used. With a *`key_num`* argument, the given key number (0 to 9) from the DES key file is used. With a *`key_str`* argument, the given key string is used to encrypt *`str`*.

The key file can be specified with the `--des-key-file` [386] server option.

The return string is a binary string where the first character is `CHAR(128 | key_num)` [794]. If an error occurs, `DES_ENCRYPT()` [866] returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *`key_num`* is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *`key_num`* value must be a number in the range from `0` to `9`. Lines in the file may be in any order. *`des_key_str`* is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()` [866].

You can tell MySQL to read new key values from the key file with the `FLUSH DES_KEY_FILE` statement. This requires the `RELOAD` [492] privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table
    > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

`DES_ENCRYPT()` [866] was added in MySQL 4.0.1.

- `ENCODE(str,pass_str)` [867]

  Encrypt *`str`* using *`pass_str`* as the password. The result is a binary string of the same length as *`str`*. To decrypt the result, use `DECODE()` [866].

  The strength of the encryption is based on how good the random generator is. It should suffice for short strings.

- `ENCRYPT(str[,salt])` [867]

  Encrypts *`str`* using the Unix `crypt()` system call and returns a binary string. The *`salt`* argument must be a string with at least two characters or the result will be `NULL`. (As of MySQL 3.22.16, *`salt`* may be longer than two characters.) If no *`salt`* argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
        -> 'VxuFAJXVARROc'
```

ENCRYPT() [867] ignores all but the first eight characters of *str*, at least on some systems. This behavior is determined by the implementation of the underlying crypt() system call.

The use of ENCRYPT() [867] with the ucs2 multi-byte character set is not recommended because the system call expects a string terminated by a zero byte.

If crypt() is not available on your system (as is the case with Windows), ENCRYPT() [867] always returns NULL.

- MD5(*str*) [868]

  Calculates an MD5 128-bit checksum for the string. The value is returned as a binary string of 32 hex digits, or NULL if the argument was NULL. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

  ```
  mysql> SELECT MD5('testing');
          -> 'ae2b1fca515949e5d54fb22b8ed95575'
  ```

  This is the "RSA Data Security, Inc. MD5 Message-Digest Algorithm."

  See the note regarding the MD5 algorithm at the beginning this section.

  MD5() [868] was added in MySQL 3.23.2.

- OLD_PASSWORD(*str*) [868]

  OLD_PASSWORD() [868] is available as of MySQL 4.1, when the implementation of PASSWORD() [868] was changed to improve security. OLD_PASSWORD() [868] returns the value of the pre-4.1 implementation of PASSWORD() [868] as a binary string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your version 4.1 MySQL server without locking them out. See Section 5.4.2.3, "Password Hashing in MySQL".

- PASSWORD(*str*) [868]

  Calculates and returns a password string from the plaintext password *str* and returns a binary string, or NULL if the argument was NULL. This is the function that is used for encrypting MySQL passwords for storage in the Password column of the user grant table.

  ```
  mysql> SELECT PASSWORD('badpwd');
          -> '7f84554057dd964b'
  ```

  PASSWORD() [868] encryption is one-way (not reversible).

  PASSWORD() [868] does not perform password encryption in the same way that Unix passwords are encrypted. See ENCRYPT() [867].

  Note

  The PASSWORD() [868] function is used by the authentication system in MySQL Server; you should *not* use it in your own applications. For that purpose, consider MD5() [868] or SHA1() [869] instead. Also see RFC 2195, section 2

(Challenge-Response Authentication Mechanism (CRAM)), for more information about handling passwords and authentication securely in your applications.

Important

Statements that invoke `PASSWORD()` [868] may be recorded in server logs or in a history file such as `~/.mysql_history`, which means that plaintext passwords may be read by anyone having read access to that information. See Section 5.4.2, “Password Security in MySQL”.

- `SHA1(str)` [869], `SHA(str)` [869]

  Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a binary string of 40 hex digits, or `NULL` if the argument was `NULL`. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. You can also use `SHA1()` [869] as a cryptographic function for storing passwords. `SHA()` [869] is synonymous with `SHA1()` [869].

```
mysql> SELECT SHA1('abc');
        -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

  `SHA1()` [869] was added in MySQL 4.0.2, and can be considered a cryptographically more secure equivalent of `MD5()` [868]. However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- `UNCOMPRESS(string_to_uncompress)` [869]

  Uncompresses a string compressed by the `COMPRESS()` [866] function. If the argument is not a compressed value, the result is `NULL`. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
        -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
        -> NULL
```

  `UNCOMPRESS()` [869] was added in MySQL 4.1.1.

- `UNCOMPRESSED_LENGTH(compressed_string)` [869]

  Returns the length that the compressed string had before being compressed.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
        -> 30
```

  `UNCOMPRESSED_LENGTH()` [869] was added in MySQL 4.1.1.

11.13 Information Functions

Table 11.17 Information Functions

Name	Description
`BENCHMARK()` [870]	Repeatedly execute an expression
`CHARSET()` [870]	Return the character set of the argument
`COERCIBILITY()` [871]	Return the collation coercibility value of the string argument

Name	Description
COLLATION() [871]	Return the collation of the string argument
CONNECTION_ID() [872]	Return the connection ID (thread ID) for the connection
CURRENT_USER(), CURRENT_USER [872]	The authenticated user name and host name
DATABASE() [872]	Return the default (current) database name
FOUND_ROWS() [872]	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
LAST_INSERT_ID() [874]	Value of the AUTOINCREMENT column for the last INSERT
SESSION_USER() [876]	Synonym for USER()
SYSTEM_USER() [876]	Synonym for USER()
USER() [876]	The user name and host name provided by the client
VERSION() [876]	Returns a string that indicates the MySQL server version

- BENCHMARK(*count*,*expr*) [870]

  The BENCHMARK() [870] function executes the expression *expr* repeatedly *count* times. It may be used to time how quickly MySQL processes the expression. The result value is always 0. The intended use is from within the mysql client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+----------------------------------------------+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
1 row in set (4.74 sec)
```

  The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute BENCHMARK() [870] several times, and to interpret the result with regard to how heavily loaded the server machine is.

  BENCHMARK() [870] is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

  - Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, BENCHMARK(10, (SELECT * FROM t)) [870] will fail if the table t has more than one column or more than one row.
  - Executing a SELECT *expr* statement *N* times differs from executing SELECT BENCHMARK(*N*, *expr*) in terms of the amount of overhead involved. The two have very different execution profiles and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation *N* times each. The latter involves only runtime evaluation *N* times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of BENCHMARK() [870] thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.

- CHARSET(*str*) [870]

  Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
        -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
        -> 'utf8'
mysql> SELECT CHARSET(USER());
        -> 'utf8'
```

`CHARSET()` [870] was added in MySQL 4.1.0.

- `COERCIBILITY(str)` [871]

  Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
```

The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
0	Explicit collation	Value with `COLLATE` clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value
3	System constant	`USER()` [876] return value
4	Coercible	Literal string
5	Ignorable	`NULL` or an expression derived from `NULL`

Before MySQL 4.1.11, the return values are shown in following table, and functions such as `USER()` [876] have a coercibility of 2.

Coercibility	Meaning	Example
0	Explicit collation	Value with `COLLATE` clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value
3	Coercible	Literal string

`COERCIBILITY()` [871] was added in MySQL 4.1.1.

- `COLLATION(str)` [871]

  Returns the collation of the string argument.

```
```

```
mysql> SELECT COLLATION('abc');
        -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
        -> 'utf8_general_ci'
```

`COLLATION()` [871] was added in MySQL 4.1.0.

- `CONNECTION_ID()` [872]

  Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

```
mysql> SELECT CONNECTION_ID();
        -> 23786
```

  `CONNECTION_ID()` [872] was added in MySQL 3.23.14.

- `CURRENT_USER` [872], `CURRENT_USER()` [872]

  Returns the user name and host name combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges.

  The value of `CURRENT_USER()` [872] can differ from the value of `USER()` [876].

```
mysql> SELECT USER();
        -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
        -> '@localhost'
```

  The example illustrates that although the client specified a user name of `davida` (as indicated by the value of the `USER()` [876] function), the server authenticated the client using an anonymous user account (as seen by the empty user name part of the `CURRENT_USER()` [872] value). One way this might occur is that there is no account listed in the grant tables for `davida`.

  `CURRENT_USER()` [872] was added in MySQL 4.0.6. As of MySQL 4.1.0, the string uses the `utf8` character set.

- `DATABASE()` [872]

  Returns the default (current) database name. As of MySQL 4.1, the string uses the `utf8` character set. If there is no default database, `DATABASE()` [872] returns `NULL` as of MySQL 4.1.1, and the empty string before that.

```
mysql> SELECT DATABASE();
        -> 'test'
```

- `FOUND_ROWS()` [872]

  A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include a `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke `FOUND_ROWS()` [872] afterward:

```
```

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

The second SELECT returns a number indicating how many rows the first SELECT would have returned had it been written without the LIMIT clause.

In the absence of the SQL_CALC_FOUND_ROWS option in the most recent successful SELECT statement, FOUND_ROWS() [872] returns the number of rows in the result set returned by that statement. If the statement includes a LIMIT clause, FOUND_ROWS() [872] returns the number of rows up to the limit. For example, FOUND_ROWS() [872] returns 10 or 60, respectively, if the statement includes LIMIT 10 or LIMIT 50, 10.

The row count available through FOUND_ROWS() [872] is transient and not intended to be available past the statement following the SELECT SQL_CALC_FOUND_ROWS statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

If you are using SELECT SQL_CALC_FOUND_ROWS, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without LIMIT, because the result set need not be sent to the client.

SQL_CALC_FOUND_ROWS and FOUND_ROWS() [872] can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using FOUND_ROWS() [872] enables you to determine how many other pages are needed for the rest of the result.

The use of SQL_CALC_FOUND_ROWS and FOUND_ROWS() [872] is more complex for UNION statements than for simple SELECT statements, because LIMIT may occur at multiple places in a UNION. It may be applied to individual SELECT statements in the UNION, or global to the UNION result as a whole.

The intent of SQL_CALC_FOUND_ROWS for UNION is that it should return the row count that would be returned without a global LIMIT. The conditions for use of SQL_CALC_FOUND_ROWS with UNION are:

- The SQL_CALC_FOUND_ROWS keyword must appear in the first SELECT of the UNION.
- The value of FOUND_ROWS() [872] is exact only if UNION ALL is used. If UNION without ALL is used, duplicate removal occurs and the value of FOUND_ROWS() [872] is only approximate.
- If no LIMIT is present in the UNION, SQL_CALC_FOUND_ROWS is ignored and returns the number of rows in the temporary table that is created to process the UNION.

Beyond the cases described here, the behavior of FOUND_ROWS() [872] is undefined (for example, its value following a SELECT statement that fails with an error).

SQL_CALC_FOUND_ROWS and FOUND_ROWS() [872] are available starting at MySQL 4.0.0.

Important

FOUND_ROWS() [872] is not replicated reliably, and should not be used with databases that are to be replicated.

- `LAST_INSERT_ID()` [874], `LAST_INSERT_ID(expr)` [874]

  `LAST_INSERT_ID()` [874] (with no argument) returns the *first* automatically generated value that was set for an `AUTO_INCREMENT` column by the *most recently executed* `INSERT` or `UPDATE` statement to affect such a column. For example, after inserting a row that generates an `AUTO_INCREMENT` value, you can get the value like this:

```
mysql> SELECT LAST_INSERT_ID();
        -> 195
```

  if a table contains an `AUTO_INCREMENT` column and `INSERT ... ON DUPLICATE KEY UPDATE` updates (rather than inserts) a row, the value of `LAST_INSERT_ID()` [874] is not meaningful. For a workaround, see Section 12.2.4.3, "`INSERT ... ON DUPLICATE KEY UPDATE` Syntax".

  The currently executing statement does not affect the value of `LAST_INSERT_ID()` [874]. Suppose that you generate an `AUTO_INCREMENT` value with one statement, and then refer to `LAST_INSERT_ID()` [874] in a multiple-row `INSERT` statement that inserts rows into a table with its own `AUTO_INCREMENT` column. The value of `LAST_INSERT_ID()` [874] will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to `LAST_INSERT_ID()` [874] and `LAST_INSERT_ID(expr)` [874], the effect is undefined.)

  If the previous statement returned an error, the value of `LAST_INSERT_ID()` [874] is undefined. For transactional tables, if the statement is rolled back due to an error, the value of `LAST_INSERT_ID()` [874] is left undefined. For manual `ROLLBACK`, the value of `LAST_INSERT_ID()` [874] is not restored to that before the transaction; it remains as it was at the point of the `ROLLBACK`.

  The ID that was generated is maintained in the server on a *per-connection basis*. This means that the value returned by the function to a given client is the first `AUTO_INCREMENT` value generated for most recent statement affecting an `AUTO_INCREMENT` column *by that client*. This value cannot be affected by other clients, even if they generate `AUTO_INCREMENT` values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

  The value of `LAST_INSERT_ID()` [874] is not changed if you set the `AUTO_INCREMENT` column of a row to a non-"magic" value (that is, a value that is not `NULL` and not `0`).

  Important

  If you insert multiple rows using a single `INSERT` statement, `LAST_INSERT_ID()` [874] returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same `INSERT` statement against some other server.

  For example:

```
mysql> USE test;
Database changed
mysql> CREATE TABLE t (
    ->   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   name VARCHAR(10) NOT NULL
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t VALUES (NULL, 'Bob');
```

```
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
+----+------+
1 row in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO t VALUES
    -> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+------+
4 rows in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                2 |
+------------------+
1 row in set (0.00 sec)
```

Although the second `INSERT` statement inserted three new rows into `t`, the ID generated for the first of these rows was `2`, and it is this value that is returned by `LAST_INSERT_ID()` [874] for the following `SELECT` statement.

If you use `INSERT IGNORE` and the row is ignored, the `AUTO_INCREMENT` counter is not incremented and `LAST_INSERT_ID()` [874] returns `0`, which reflects that no row was inserted. (Before MySQL 4.1, the `AUTO_INCREMENT` counter is still incremented and `LAST_INSERT_ID()` [874] returns the new value.)

If *`expr`* is given as an argument to `LAST_INSERT_ID()` [874], the value of the argument is returned by the function and is remembered as the next value to be returned by `LAST_INSERT_ID()` [874]. This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

   ```
   mysql> CREATE TABLE sequence (id INT NOT NULL);
   mysql> INSERT INTO sequence VALUES (0);
   ```

2. Use the table to generate sequence numbers like this:

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

The `UPDATE` statement increments the sequence counter and causes the next call to `LAST_INSERT_ID()` [874] to return the updated value. The `SELECT` statement retrieves that value. The `mysql_insert_id()` C API function can also be used to get the value. See Section 17.6.6.35, “`mysql_insert_id()`”.

You can generate sequences without calling `LAST_INSERT_ID()` [874], but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the `UPDATE` statement and get their own sequence value with the `SELECT` statement (or `mysql_insert_id()`), without affecting or being affected by other clients that generate their own sequence values.

Note that `mysql_insert_id()` is only updated after `INSERT` and `UPDATE` statements, so you cannot use the C API function to retrieve the value for `LAST_INSERT_ID(`*expr*`)` [874] after executing other SQL statements like `SELECT` or `SET`.

- `SESSION_USER()` [876]

  `SESSION_USER()` [876] is a synonym for `USER()` [876].

- `SYSTEM_USER()` [876]

  `SYSTEM_USER()` [876] is a synonym for `USER()` [876].

- `USER()` [876]

  Returns the current MySQL user name and host name.

```
mysql> SELECT USER();
        -> 'davida@localhost'
```

The value indicates the user name you specified when connecting to the server, and the client host from which you connected. The value can be different from that of `CURRENT_USER()` [872].

Prior to MySQL 3.22.11, the function value does not include the client host name. You can extract only the user name part, regardless of whether the value includes a host name part, like this:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
        -> 'davida'
```

As of MySQL 4.1, `USER()` [876] returns a value in the `utf8` character set, so you should also make sure that the `'@'` string literal is interpreted in that character set:

```
mysql> SELECT SUBSTRING_INDEX(USER(),_utf8'@',1);
        -> 'davida'
```

- `VERSION()` [876]

  Returns a string that indicates the MySQL server version. As of MySQL 4.1, the string has the `utf8` character set.

```
mysql> SELECT VERSION();
        -> '4.1.25-standard'
```

Note that if your version string ends with `-log` this means that logging is enabled.

11.14 Miscellaneous Functions

Table 11.18 Miscellaneous Functions

Name	Description
`DEFAULT()` [877]	Return the default value for a table column
`GET_LOCK()` [877]	Get a named lock
`INET_ATON()` [878]	Return the numeric value of an IP address
`INET_NTOA()` [878]	Return the IP address from a numeric value
`IS_FREE_LOCK()` [879]	Checks whether the named lock is free
`IS_USED_LOCK()` [879]	Checks whether the named lock is in use. Return connection identifier if true.
`MASTER_POS_WAIT()` [879]	Block until the slave has read and applied all updates up to the specified position
`RAND()` [822]	Return a random floating-point value
`RELEASE_LOCK()` [879]	Releases the named lock
`UUID()` [879]	Return a Universal Unique Identifier (UUID)
`VALUES()` [880]	Defines the values to be used during an INSERT

- `DEFAULT(col_name)` [877]

  Returns the default value for a table column.

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

  `DEFAULT()` [877] was added in MySQL 4.1.0.

- `FORMAT(X,D)` [796]

  Formats the number *`X`* to a format like `'#,###,###.##'`, rounded to *`D`* decimal places, and returns the result as a string. For details, see Section 11.5, “String Functions”.

- `GET_LOCK(str,timeout)` [877]

  Tries to obtain a lock with a name given by the string *`str`*, using a timeout of *`timeout`* seconds. Returns `1` if the lock was obtained successfully, `0` if the attempt timed out (for example, because another client has previously locked the name), or `NULL` if an error occurred (such as running out of memory or the thread was killed with `mysqladmin kill`). If you have a lock obtained with `GET_LOCK()` [877], it is released when you execute `RELEASE_LOCK()` [879], execute a new `GET_LOCK()` [877], or your connection terminates (either normally or abnormally). Locks obtained with `GET_LOCK()` [877] do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

  This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, `GET_LOCK()` [877] blocks any request by another client for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of

this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *`db_name.str`* or *`app_name.str`*.

```
mysql> SELECT GET_LOCK('lock1',10);
        -> 1
mysql> SELECT IS_FREE_LOCK('lock2');
        -> 1
mysql> SELECT GET_LOCK('lock2',10);
        -> 1
mysql> SELECT RELEASE_LOCK('lock2');
        -> 1
mysql> SELECT RELEASE_LOCK('lock1');
        -> NULL
```

The second `RELEASE_LOCK()` [879] call returns `NULL` because the lock `'lock1'` was automatically released by the second `GET_LOCK()` [877] call.

If multiple clients are waiting for a lock, the order in which they will acquire it is undefined and depends on factors such as the thread library in use. In particular, applications should not assume that clients will acquire the lock in the same order that they issued the lock requests.

Note

If a client attempts to acquire a lock that is already held by another client, it blocks according to the *`timeout`* argument. If the blocked client terminates, its thread does not die until the lock request times out. This is a known bug (fixed in MySQL 5.5).

- `INET_ATON(`*`expr`*`)` [878]

Given the dotted-quad representation of a network address as a string, returns an integer that represents the numeric value of the address. Addresses may be 4- or 8-byte addresses.

```
mysql> SELECT INET_ATON('209.207.224.40');
        -> 3520061480
```

The generated number is always in network byte order. For the example just shown, the number is calculated as $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$.

As of MySQL 4.1.2, `INET_ATON()` [878] also understands short-form IP addresses:

```
mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');
        -> 2130706433, 2130706433
```

Note

When storing values generated by `INET_ATON()` [878], it is recommended that you use an `INT UNSIGNED` column. If you use a (signed) `INT` column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See Section 10.2.5, “Out-of-Range and Overflow Handling”.

`INET_ATON()` [878] was added in MySQL 3.23.15.

- `INET_NTOA(`*`expr`*`)` [878]

Given a numeric network address in network byte order (4 or 8 byte), returns the dotted-quad representation of the address as a binary string.

```
mysql> SELECT INET_NTOA(3520061480);
       -> '209.207.224.40'
```

`INET_NTOA()` [878] was added in MySQL 3.23.15.

- `IS_FREE_LOCK(str)` [879]

  Checks whether the lock named *`str`* is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument).

  `IS_FREE_LOCK()` [879] was added in MySQL 4.0.2.

- `IS_USED_LOCK(str)` [879]

  Checks whether the lock named *`str`* is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns `NULL`.

  `IS_USED_LOCK()` [879] was added in MySQL 4.1.0.

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])` [879]

  This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns `NULL` if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns `-1` if the timeout has been exceeded. If the slave SQL thread stops while `MASTER_POS_WAIT()` [879] is waiting, the function returns `NULL`. If the slave is past the specified position, the function returns immediately.

  If a *`timeout`* value is specified, `MASTER_POS_WAIT()` [879] stops waiting when *`timeout`* seconds have elapsed. *`timeout`* must be greater than 0; a zero or negative *`timeout`* means no timeout.

  `MASTER_POS_WAIT()` [879] was added in MySQL 3.23.32. The *`timeout`* argument was added in 4.0.10.

- `RELEASE_LOCK(str)` [879]

  Releases the lock named by the string *`str`* that was obtained with `GET_LOCK()` [877]. Returns `1` if the lock was released, `0` if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` [877] or if it has previously been released.

  The `DO` statement is convenient to use with `RELEASE_LOCK()` [879]. See Section 12.2.2, “DO Syntax”.

- `UUID()` [879]

  Returns a Universal Unique Identifier (UUID) generated according to “DCE 1.1: Remote Procedure Call” (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 (Document Number C706, http://www.opengroup.org/public/pubs/catalog/c706.htm).

  A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` [879] are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

  A UUID is a 128-bit number represented by a `utf8` string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

- The first three numbers are generated from a timestamp.
- The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

  Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();
        -> '6ccd780c-baba-1026-9564-0040f4311e29'
```

Warning

The `UUID()` [879] function returns a string using the character set defined by the `character_set_server` [408] parameter. If you are using UUID values in your tables and these columns are indexed the character set of your column or table should match the character set used when the `UUID()` [879] was called. If you do not use the same character set for the column and the UUID value, the indexes on those columns will not be used, which may lead to a reduction in performance and locked tables during operations as the table is searched sequentially for the value.

You can convert between different character sets when using UUID-based strings using the `CONVERT()` [859] function.

Note

`UUID()` [879] does not work with statement-based replication.

`UUID()` [879] was added in MySQL 4.1.2.

- `VALUES(col_name)` [880]

In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` [880] function is meaningful only in the `ON DUPLICATE KEY UPDATE` clause of `INSERT` statements and returns `NULL` otherwise. See Section 12.2.4.3, “`INSERT ... ON DUPLICATE KEY UPDATE` Syntax”.

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

`VALUES()` [880] was added in MySQL 4.1.1.

11.15 Functions and Modifiers for Use with GROUP BY Clauses

11.15.1 GROUP BY (Aggregate) Functions

Table 11.19 Aggregate (GROUP BY) Functions

Name	Description
AVG() [881]	Return the average value of the argument
BIT_AND() [882]	Return bitwise and
BIT_OR() [882]	Return bitwise or
BIT_XOR() [882]	Return bitwise xor
COUNT(DISTINCT) [882]	Return the count of a number of different values
COUNT() [882]	Return a count of the number of rows returned
GROUP_CONCAT() [883]	Return a concatenated string
MAX() [884]	Return the maximum value
MIN() [884]	Return the minimum value
STD() [884]	Return the population standard deviation
STDDEV() [884]	Return the population standard deviation
SUM() [884]	Return the sum
VARIANCE() [884]	Return the population standard variance

This section describes group (aggregate) functions that operate on sets of values. Unless otherwise stated, group functions ignore NULL values.

If you use a group function in a statement containing no GROUP BY clause, it is equivalent to grouping on all rows. For more information, see Section 11.15.3, “GROUP BY and HAVING with Hidden Columns”.

For numeric arguments, the variance, standard deviation, SUM() [884], and AVG() [881] functions return a DOUBLE value.

The SUM() [884] and AVG() [881] aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, you can convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Functions such as SUM() [884] or AVG() [881] that expect a numeric argument cast the argument to a number if necessary. For SET or ENUM values, the cast operation causes the underlying numeric value to be used.

- Returns the average value of *expr*.

  AVG() [881] returns NULL if there were no matching rows.

```
mysql> SELECT student_name, AVG(test_score)
    ->        FROM student
```

```
    ->        GROUP BY student_name;
```

- `BIT_AND(expr)` [882]

  Returns the bitwise `AND` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  As of MySQL 4.0.17, this function returns `18446744073709551615` if there were no matching rows. (This is the value of an unsigned `BIGINT` value with all bits set to 1.) Before 4.0.17, the function returns `-1` if there were no matching rows.

- `BIT_OR(expr)` [882]

  Returns the bitwise `OR` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  This function returns `0` if there were no matching rows.

- `BIT_XOR(expr)` [882]

  Returns the bitwise `XOR` [788] of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  This function returns `0` if there were no matching rows.

  This function is available as of MySQL 4.1.1.

- `COUNT(expr)` [882]

  Returns a count of the number of non-`NULL` values of *`expr`* in the rows retrieved by a `SELECT` statement. The result is a `BIGINT` value.

  `COUNT()` [882] returns `0` if there were no matching rows.

```
mysql> SELECT student.student_name,COUNT(*)
    ->        FROM student,course
    ->        WHERE student.student_id=course.student_id
    ->        GROUP BY student_name;
```

  `COUNT(*)` [882] is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain `NULL` values.

  `COUNT(*)` [882] is optimized to return very quickly if the `SELECT` retrieves from one table, no other columns are retrieved, and there is no `WHERE` clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

  This optimization applies only to `MyISAM` and `ISAM` tables only, because an exact row count is stored for these storage engines and can be accessed very quickly. For transactional storage engines such as `InnoDB` and `BDB`, storing an exact row count is more problematic because multiple transactions may be occurring, each of which may affect the count.

- `COUNT(DISTINCT expr,[expr...])` [882]

  Returns a count of the number of rows with different non-`NULL` *`expr`* values.

  `COUNT(DISTINCT)` [882] returns `0` if there were no matching rows.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

In MySQL, you can obtain the number of distinct expression combinations that do not contain NULL by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside COUNT(DISTINCT ...) [882].

COUNT(DISTINCT ...) [882] was added in MySQL 3.23.2.

- GROUP_CONCAT(*expr*) [883]

  This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(test_score)
    ->     FROM student
    ->     GROUP BY student_name;
```

Or:

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(DISTINCT test_score
    ->               ORDER BY test_score DESC SEPARATOR ' ')
    ->     FROM student
    ->     GROUP BY student_name;
```

In MySQL, you can get the concatenated values of expression combinations. To eliminate duplicate values, use the DISTINCT clause. To sort values in the result, use the ORDER BY clause. To sort in reverse order, add the DESC (descending) keyword to the name of the column you are sorting by in the ORDER BY clause. The default is ascending order; this may be specified explicitly using the ASC keyword. The default separator between values in a group is comma (“,”). To specify a separator explicitly, use SEPARATOR followed by the string value that should be inserted between group values. To eliminate the separator altogether, specify SEPARATOR ''.

The result is truncated to the maximum length that is given by the group_concat_max_len [412] system variable, which has a default value of 1024. The value can be set higher, although the effective maximum length of the return value is constrained by the value of max_allowed_packet [418]. The syntax to change the value of group_concat_max_len [412] at runtime is as follows, where *val* is an unsigned integer:

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

The return value is a nonbinary or binary string, depending on whether the arguments are nonbinary or binary strings. The result type is TEXT or BLOB unless group_concat_max_len [412] is less than or equal to 255, in which case the result type is CHAR or BINARY. (Prior to MySQL 4.1.19, GROUP_CONCAT() [883] returned TEXT or BLOB group_concat_max_len [412] greater than 255 only if the query included an ORDER BY clause.)

GROUP_CONCAT() [883] was added in MySQL 4.1.

Note: Before MySQL 4.1.6, there are some small limitations with GROUP_CONCAT() [883] for BLOB and TEXT values when it comes to using DISTINCT together with ORDER BY. To work around this limitation, use MID(*expr*,1,255) [799] instead.

See also CONCAT() [795] and CONCAT_WS() [795]: Section 11.5, “String Functions”.

- MAX(*expr*) [884]

  Returns the maximum value of *expr*. MAX() [884] may take a string argument; in such cases, it returns the maximum string value. See Section 7.4.3, “How MySQL Uses Indexes”.

  MAX() [884] returns NULL if there were no matching rows.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

  For MAX() [884], MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set. This differs from how ORDER BY compares them. This is expected to be rectified in a future MySQL release.

- MIN(*expr*) [884]

  Returns the minimum value of *expr*. MIN() [884] may take a string argument; in such cases, it returns the minimum string value. See Section 7.4.3, “How MySQL Uses Indexes”.

  MIN() [884] returns NULL if there were no matching rows.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

  For MIN() [884], MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set. This differs from how ORDER BY compares them. This is expected to be rectified in a future MySQL release.

- STD(*expr*) [884]

  Returns the population standard deviation of *expr*. This is an extension to standard SQL.

  This function returns NULL if there were no matching rows.

- STDDEV(*expr*) [884]

  Returns the population standard deviation of *expr*. This function is provided for compatibility with Oracle.

  This function returns NULL if there were no matching rows.

- SUM(*expr*) [884]

  Returns the sum of *expr*. If the return set has no rows, SUM() [884] returns NULL.

  SUM() [884] returns NULL if there were no matching rows.

- VARIANCE(*expr*) [884]

Returns the population standard variance of *expr*. This is an extension to standard SQL, available in MySQL 4.1 or later.

VARIANCE() [884] returns NULL if there were no matching rows.

11.15.2 GROUP BY Modifiers

As of MySQL 4.1.1, the GROUP BY clause permits a WITH ROLLUP modifier that causes extra rows to be added to the summary output. These rows represent higher-level (or super-aggregate) summary operations. ROLLUP thus enables you to answer questions at multiple levels of analysis with a single query. It can be used, for example, to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a table named sales has year, country, product, and profit columns for recording sales profitability:

```
CREATE TABLE sales
(
    year    INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit  INT
);
```

The table's contents can be summarized per year with a simple GROUP BY like this:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```

This output shows the total profit for each year, but if you also want to determine the total profit summed over all years, you must add up the individual values yourself or run an additional query.

Or you can use ROLLUP, which provides both levels of analysis with a single query. Adding a WITH ROLLUP modifier to the GROUP BY clause causes the query to produce another row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
| NULL |        7535 |
+------+-------------+
```

The grand total super-aggregate line is identified by the value NULL in the year column.

ROLLUP has a more complex effect when there are multiple GROUP BY columns. In this case, each time there is a “break” (change in value) in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without ROLLUP, a summary on the sales table based on year, country, and product might look like this:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2001 | Finland | Phone      |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
+------+---------+------------+-------------+
```

The output indicates summary values only at the year/country/product level of analysis. When `ROLLUP` is added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | India   | NULL       |        1350 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2000 | USA     | NULL       |        1575 |
| 2000 | NULL    | NULL       |        4525 |
| 2001 | Finland | Phone      |          10 |
| 2001 | Finland | NULL       |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
| 2001 | USA     | NULL       |        3000 |
| 2001 | NULL    | NULL       |        3010 |
| NULL | NULL    | NULL       |        7535 |
+------+---------+------------+-------------+
```

For this query, adding `ROLLUP` causes the output to include summary information at four levels of analysis, not just one. Here is how to interpret the `ROLLUP` output:

- Following each set of product rows for a given year and country, an extra summary row is produced showing the total for all products. These rows have the `product` column set to `NULL`.
- Following each set of rows for a given year, an extra summary row is produced showing the total for all countries and products. These rows have the `country` and `products` columns set to `NULL`.
- Finally, following all other rows, an extra summary row is produced showing the grand total for all years, countries, and products. This row has the `year`, `country`, and `products` columns set to `NULL`.

Other Considerations When using `ROLLUP`

The following items list some behaviors specific to the MySQL implementation of `ROLLUP`.

When you use ROLLUP, you cannot also use an ORDER BY clause to sort the results. In other words, ROLLUP and ORDER BY are mutually exclusive. However, you still have some control over sort order. GROUP BY in MySQL sorts results, and you can use explicit ASC and DESC keywords with columns named in the GROUP BY list to specify sort order for individual columns. (The higher-level summary rows added by ROLLUP still appear after the rows from which they are calculated, regardless of the sort order.)

LIMIT can be used to restrict the number of rows returned to the client. LIMIT is applied after ROLLUP, so the limit applies against the extra rows added by ROLLUP. For example:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP
    -> LIMIT 5;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
+------+---------+------------+-------------+
```

Using LIMIT with ROLLUP may produce results that are more difficult to interpret, because you have less context for understanding the super-aggregate rows.

The NULL indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the GROUP BY clause following the leftmost one that has changed value. For any column in the result set with a name that is a lexical match to any of those names, its value is set to NULL. (If you specify grouping columns by column number, the server identifies which columns to set to NULL by number.)

Because the NULL values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as NULL values within the query itself. For example, you cannot add HAVING product IS NULL to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the NULL values do appear as NULL on the client side and can be tested as such using any MySQL client programming interface.

MySQL permits a column that does not appear in the GROUP BY list to be named in the select list. In this case, the server is free to choose any value from this nonaggregated column in summary rows, and this includes the extra rows added by WITH ROLLUP. For example, in the following query, country is a nonaggregated column that does not appear in the GROUP BY list and values chosen for this column are indeterminate:

```
mysql> SELECT year, country, SUM(profit)
    -> FROM sales GROUP BY year WITH ROLLUP;
+------+---------+-------------+
| year | country | SUM(profit) |
+------+---------+-------------+
| 2000 | India   |        4525 |
| 2001 | USA     |        3010 |
| NULL | USA     |        7535 |
+------+---------+-------------+
```

This behavior occurs if the ONLY_FULL_GROUP_BY [460] SQL mode is not enabled. If that mode is enabled, the server rejects the query as illegal because country is not listed in the GROUP BY clause. For more information about nonaggregated columns and GROUP BY, see MySQL Extensions to GROUP BY.

11.15.3 GROUP BY and HAVING with Hidden Columns

MySQL extends the use of GROUP BY so that you can use nonaggregated columns or calculations in the select list that do not appear in the GROUP BY clause. You can use this feature to get better performance by avoiding unnecessary column sorting and grouping. For example, you need not group on customer.name in the following query:

```
SELECT order.custid, customer.name, MAX(payments)
  FROM order,customer
  WHERE order.custid = customer.custid
  GROUP BY order.custid;
```

In standard SQL, you would have to add customer.name to the GROUP BY clause. In MySQL, the name is redundant.

When using this feature, all rows in each group should have the same values for the columns that are ommitted from the GROUP BY part. The server is free to return any value from the group, so the results are indeterminate unless all values are the same.

A similar MySQL extension applies to the HAVING clause. The SQL standard does not permit the HAVING clause to name any column not found in the GROUP BY clause if it is not enclosed in an aggregate function. MySQL permits the use of such columns to simplify calculations. This extension assumes that the nongrouped columns will have the same group-wise values. Otherwise, the result is indeterminate.

If the ONLY_FULL_GROUP_BY [460] SQL mode is enabled, the MySQL extension to GROUP BY does not apply to the SELECT. That is, columns not named in the GROUP BY clause cannot be used in the SELECT list if not used in an aggregate function.

The select list extension also applies to ORDER BY. That is, you can use nonaggregated columns or calculations in the ORDER BY clause that do not appear in the GROUP BY clause. This extension does not apply if the ONLY_FULL_GROUP_BY [460] SQL mode is enabled.

In some cases, you can use MIN() [884] and MAX() [884] to obtain a specific column value even if it is not unique. The following gives the value of column from the row containing the smallest value in the sort column:

```
SUBSTR(MIN(CONCAT(RPAD(sort,6,' '),column)),7)
```

See Section 3.6.4, “The Rows Holding the Group-wise Maximum of a Certain Column”.

If you are trying to follow standard SQL, you cannot use expressions in GROUP BY clauses. As a workaround, use an alias for the expression:

```
SELECT id, FLOOR(value/100) AS val
  FROM tbl_name
  GROUP BY id, val;
```

MySQL permits expressions in GROUP BY clauses, so the alias is unnecessary:

```
SELECT id, FLOOR(value/100)
  FROM tbl_name
  GROUP BY id, FLOOR(value/100);
```

Chapter 12 SQL Statement Syntax

Table of Contents

This chapter describes the syntax for the SQL statements supported in MySQL versions 4.1 and earlier.

12.1 Data Definition Statements

12.1.1 ALTER DATABASE Syntax

```
ALTER DATABASE [db_name]
    alter_specification ...

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

`ALTER DATABASE` enables you to change the overall characteristics of a database. These characteristics are stored in the `db.opt` file in the database directory. To use `ALTER DATABASE`, you need the `ALTER` [491] privilege on the database.

The `CHARACTER SET` clause changes the default database character set. The `COLLATE` clause changes the default database collation. Section 9.1, “Character Set Support”, discusses character set and collation names.

Beginning with MySQL 4.1.0, you can see what character sets and collations are available using, respectively, the `SHOW CHARACTER SET` and `SHOW COLLATION` statements. See Section 12.4.5.3, “`SHOW CHARACTER SET` Syntax”, and Section 12.4.5.4, “`SHOW COLLATION` Syntax”, for more information.

`ALTER DATABASE` was added in MySQL 4.1.1. Beginning with MySQL 4.1.8, the database name can be omitted, in which case the statement applies to the default database.

12.1.2 ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name ]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX|KEY} [index_name]
        [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX|KEY] [index_name]
        [index_type] (index_col_name,...)
  | ADD [FULLTEXT|SPATIAL] [INDEX|KEY] [index_name]
        (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        reference_definition
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP {INDEX|KEY} index_name
  | DROP FOREIGN KEY fk_symbol
```

```
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name [, col_name] ...
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

table_options:
    table_option [[,] table_option] ...  (see CREATE TABLE options)
```

ALTER TABLE enables you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and type of the table.

The syntax for many of the permissible alterations is similar to clauses of the CREATE TABLE statement. See Section 12.1.5, “CREATE TABLE Syntax”, for more information.

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. In MySQL 4.1 and up, these warnings can be displayed with SHOW WARNINGS. See Section 12.4.5.26, “SHOW WARNINGS Syntax”.

If you use ALTER TABLE to change a column specification but DESCRIBE *tbl_name* indicates that your column was not changed, it is possible that MySQL ignored your modification for one of the reasons described in Section 12.1.5.2, “Silent Column Specification Changes”. For example, if you try to change a VARCHAR column to CHAR, MySQL still uses VARCHAR if the table contains other variable-length columns.

In most cases, ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, and then the original table is deleted and the new one is renamed. While ALTER TABLE is executing, the original table is readable by other sessions. Updates and writes to the table are stalled until the new table is ready, and then are automatically redirected to the new table without any failed updates. The temporary table is created in the database directory of the new table. This can be different from the database directory of the original table if ALTER TABLE is renaming the table to a different database.

If you use ALTER TABLE *tbl_name* RENAME TO *new_tbl_name* without any other options, MySQL simply renames any files that correspond to the table *tbl_name*. (You can also use the RENAME TABLE statement to rename tables. See Section 12.1.9, “RENAME TABLE Syntax”.) Any privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

If you use any option to ALTER TABLE other than RENAME, MySQL always creates a temporary table, even if the data wouldn't strictly need to be copied (such as when you change the name of a column). For MyISAM tables, you can speed up the index re-creation operation (which is the slowest part of the alteration process) by setting the myisam_sort_buffer_size [421] system variable to a high value.

For information on troubleshooting ALTER TABLE, see Section B.5.7.1, “Problems with ALTER TABLE”.

- To use ALTER TABLE, you need ALTER [491], INSERT [492], and CREATE [491] privileges for the table.
- IGNORE is a MySQL extension to standard SQL. It controls how ALTER TABLE works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If IGNORE

is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If IGNORE is specified, only the first row is used of rows with duplicates on a unique key, The other conflicting rows are deleted. Incorrect values are truncated to the closest matching acceptable value.

- *table_option* signifies a table option of the kind that can be used in the CREATE TABLE statement, such as ENGINE, AUTO_INCREMENT, or AVG_ROW_LENGTH. (Section 12.1.5, "CREATE TABLE Syntax", lists all table options.) However, ALTER TABLE ignores the DATA DIRECTORY and INDEX DIRECTORY table options.

  For example, to convert a table to be an InnoDB table, use this statement:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

  To change the value of the AUTO_INCREMENT counter to be used for new rows, do this:

```
ALTER TABLE t2 AUTO_INCREMENT = value;
```

  You cannot reset the counter to a value less than or equal to any that have already been used. For MyISAM, if the value is less than or equal to the maximum value currently in the AUTO_INCREMENT column, the value is reset to the current maximum plus one. For InnoDB, you can use ALTER TABLE ... AUTO_INCREMENT = *value* as of MySQL 4.1.12, but *if the value is less than the current maximum value in the column, no error occurs and the current sequence value is not changed.*

- You can issue multiple ADD, ALTER, DROP, and CHANGE clauses in a single ALTER TABLE statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per ALTER TABLE statement. For example, to drop multiple columns in a single statement, do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- CHANGE *col_name*, DROP *col_name*, and DROP INDEX are MySQL extensions to standard SQL.
- MODIFY is an Oracle extension to ALTER TABLE.
- The word COLUMN is optional and can be omitted.
- *column_definition* clauses use the same syntax for ADD and CHANGE as for CREATE TABLE. See Section 12.1.5, "CREATE TABLE Syntax".
- You can rename a column using a CHANGE *old_col_name new_col_name column_definition* clause. To do so, specify the old and new column names and the definition that the column currently has. For example, to rename an INTEGER column from a to b, you can do this:

```
ALTER TABLE t1 CHANGE a b INTEGER;
```

  If you want to change a column's type but not the name, CHANGE syntax still requires an old and new column name, even if they are the same. For example:

```
ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

  However, as of MySQL 3.22.16a, you can also use MODIFY to change a column's type without renaming it:

```
ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

When you use CHANGE or MODIFY, *column_definition* must include the data type and all attributes that should apply to the new column, other than index attributes such as PRIMARY KEY or UNIQUE. Attributes present in the original definition but not specified for the new definition are not carried forward. Suppose that a column col1 is defined as INT UNSIGNED DEFAULT 1 COMMENT 'my column' and you modify the column as follows:

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

The resulting column will be defined as BIGINT, but will not include the attributes UNSIGNED DEFAULT 1 COMMENT 'my column'. To retain them, the statement should be:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
```

- When you change a data type using CHANGE or MODIFY, MySQL tries to convert existing column values to the new type as well as possible.

  Warning

  This conversion may result in alteration of data. For example, if you shorten a string column, values may be truncated.

- In MySQL 3.22 or later, to add a column at a specific position within a table row, use FIRST or AFTER *col_name*. The default is to add the column last. From MySQL 4.0.1 on, you can also use FIRST and AFTER in CHANGE or MODIFY operations to reorder columns within a table.

- ALTER ... SET DEFAULT or ALTER ... DROP DEFAULT specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be NULL, the new default is NULL. If the column cannot be NULL, MySQL assigns a default value as described in Section 10.1.4, “Data Type Default Values”.

- DROP INDEX removes an index. This is a MySQL extension to standard SQL. See Section 12.1.7, “DROP INDEX Syntax”. If you are unsure of the index name, use SHOW INDEX FROM *tbl_name*.

- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well. If you use CHANGE or MODIFY to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use DROP TABLE instead.

- DROP PRIMARY KEY drops the primary key. If there is no primary key, an error occurs. (Prior to MySQL 4.1.2, if no primary key exists, DROP PRIMARY KEY drops the first UNIQUE index in the table. MySQL marks the first UNIQUE key as the PRIMARY KEY if no PRIMARY KEY was specified explicitly.)

  If you add a UNIQUE INDEX or PRIMARY KEY to a table, it is stored before any nonunique index so that MySQL can detect duplicate keys as early as possible.

- From MySQL 4.1.0 on, some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING *type_name*. For details about USING, see Section 12.1.4, “CREATE INDEX Syntax”.

- ORDER BY enables you to create the new table with the rows in a specific order. Note that the table does not remain in this order after inserts and deletes. This option is useful primarily when you know that you are mostly to query the rows in a certain order most of the time. By using this option after major

changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

ORDER BY syntax permits one or more column names to be specified for sorting, each of which optionally can be followed by ASC or DESC to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are permitted as sort criteria; arbitrary expressions are not permitted.

ORDER BY does not make sense for InnoDB tables that contain a user-defined clustered index (PRIMARY KEY or NOT NULL UNIQUE index). InnoDB always orders table rows according to such an index if one is present. The same is true for BDB tables that contain a user-defined PRIMARY KEY.

- If you use ALTER TABLE on a MyISAM table, all nonunique indexes are created in a separate batch (as for REPAIR TABLE). This should make ALTER TABLE much faster when you have many indexes.

  As of MySQL 4.0, this feature can be activated explicitly for a MyISAM table. ALTER TABLE ... DISABLE KEYS tells MySQL to stop updating nonunique indexes. ALTER TABLE ... ENABLE KEYS then should be used to re-create missing indexes. MySQL does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using ALTER TABLE ... DISABLE KEYS requires the INDEX [491] privilege in addition to the privileges mentioned earlier.

  While the nonunique indexes are disabled, they are ignored for statements such as SELECT and EXPLAIN that otherwise would use them.

- If ALTER TABLE for an InnoDB table results in changes to column values (for example, because a column is truncated), InnoDB's FOREIGN KEY constraint checks do not notice possible violations caused by changing the values.

- The FOREIGN KEY and REFERENCES clauses are supported by the InnoDB storage engine, which implements ADD [CONSTRAINT [*symbol*]] FOREIGN KEY (...) REFERENCES ... (...). See Section 13.2.5.4, “FOREIGN KEY Constraints”. For other storage engines, the clauses are parsed but ignored. The CHECK clause is parsed but ignored by all storage engines. See Section 12.1.5, “CREATE TABLE Syntax”. The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See Section 1.9.5, “MySQL Differences from Standard SQL”.

  Important

  The inline REFERENCES specifications where the references are defined as part of the column specification are silently ignored by InnoDB. InnoDB only accepts REFERENCES clauses defined as part of a separate FOREIGN KEY specification.

- Starting from MySQL 4.0.13, InnoDB supports the use of ALTER TABLE to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

  For more information, see Section 13.2.5.4, “FOREIGN KEY Constraints”.

- You cannot add a foreign key and drop a foreign key in separate clauses of a single ALTER TABLE statement. You must use separate statements.

- For an InnoDB table that is created with its own tablespace in an .ibd file, that file can be discarded and imported. To discard the .ibd file, use this statement:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

This deletes the current `.ibd` file, so be sure that you have a backup first. Attempting to access the table while the tablespace file is discarded results in an error.

To import the backup `.ibd` file back into the table, copy it into the database directory, and then issue this statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

The tablespace file must have been created on the server into which it is imported later.

See Section 13.2.3.1, “Using Per-Table Tablespaces”.

- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- From MySQL 4.1.2 on, if you want to change the table default character set and all character columns (`CHAR`, `VARCHAR`, `TEXT`) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

This is useful, for example, after upgrading from MySQL 4.0.x to 4.1.x. See Section 9.1.11, “Upgrading Character Sets from MySQL 4.0”.

If you specify `CONVERT TO CHARACTER SET binary`, the `CHAR`, `VARCHAR`, and `TEXT` columns are converted to their corresponding binary string types (`BINARY`, `VARBINARY`, `BLOB`). This means that the columns no longer will have a character set and a subsequent `CONVERT TO` operation will not apply to them.

If *`charset_name`* is `DEFAULT`, the database character set is used.

Warning

The `CONVERT TO` operation converts column values between the character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8`). In this case, you have to do the following for each such column:

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

The reason this works is that there is no conversion when you convert to or from `BLOB` columns.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word `DEFAULT` is optional. The default character set is the character set that is used if you do not specify the character set for columns that you add to a table later (for example, with `ALTER TABLE ... ADD column`).

Warning

From MySQL 4.1.2 and up, `ALTER TABLE ... DEFAULT CHARACTER SET` and `ALTER TABLE ... CHARACTER SET` are equivalent and change only the default table character set. In MySQL 4.1 releases before 4.1.2, `ALTER TABLE ... DEFAULT CHARACTER SET` changes the default character set, but `ALTER TABLE ... CHARACTER SET` (without `DEFAULT`) changes the default character set *and also converts all columns to the new character set*.

With the `mysql_info()` C API function, you can find out how many rows were copied, and (when `IGNORE` is used) how many rows were deleted due to duplication of unique key values. See Section 17.6.6.33, “`mysql_info()`”.

`ALTER TABLE` Examples

Begin with a table `t1` that is created as shown here:

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column `d` and a `UNIQUE` index on column `a`:

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

To remove column `c`:

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
  ADD PRIMARY KEY (c);
```

Note that we indexed `c` (as a `PRIMARY KEY`) because `AUTO_INCREMENT` columns must be indexed, and also that we declare `c` as `NOT NULL` because primary key columns cannot be `NULL`.

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers automatically. For `MyISAM` tables, you can set the first sequence number by executing `SET`

INSERT_ID=*value* before ALTER TABLE or by using the AUTO_INCREMENT=*value* table option. See Section 5.1.3, “Server System Variables”.

With MyISAM tables, if you do not change the AUTO_INCREMENT column, the sequence number is not affected. If you drop an AUTO_INCREMENT column and then add another AUTO_INCREMENT column, the numbers are resequenced beginning with 1.

When replication is used, adding an AUTO_INCREMENT column to a table might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an AUTO_INCREMENT number. Assuming that you want to add an AUTO_INCREMENT column to the table t1, the following statements produce a new table t2 identical to t1 but with an AUTO_INCREMENT column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table t1 has columns col1 and col2.

This set of statements will also produce a new table t2 identical to t1, with the addition of an AUTO_INCREMENT column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both master and slave, *all* columns of t1 must be referenced in the ORDER BY clause.

Regardless of the method used to create and populate the copy having the AUTO_INCREMENT column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

12.1.3 CREATE DATABASE Syntax

```
CREATE DATABASE [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE [491] privilege for the database.

An error occurs if the database exists and you did not specify IF NOT EXISTS.

As of MySQL 4.1.1, *create_specification* options specify database characteristics. Database characteristics are stored in the db.opt file in the database directory. The CHARACTER SET clause

specifies the default database character set. The COLLATE clause specifies the default database collation. Section 9.1, “Character Set Support”, discusses character set and collation names.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the CREATE DATABASE statement only creates a directory under the MySQL data directory (and the db.opt file, for MySQL 4.1.1 and up). Rules for permissible database names are given in Section 8.2, “Database, Table, Index, Column, and Alias Names”.

If you manually create a directory under the data directory (for example, with mkdir), the server considers it a database directory and it shows up in the output of SHOW DATABASES.

You can also use the mysqladmin program to create databases. See Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”.

12.1.4 CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}
```

In MySQL 3.22 or later, CREATE INDEX is mapped to an ALTER TABLE statement to create indexes. See Section 12.1.2, “ALTER TABLE Syntax”. The CREATE INDEX statement does not do anything prior to MySQL 3.22. For more information about indexes, see Section 7.4.3, “How MySQL Uses Indexes”.

Normally, you create all indexes on a table at the time the table itself is created with CREATE TABLE. See Section 12.1.5, “CREATE TABLE Syntax”. CREATE INDEX enables you to add indexes to existing tables.

A column list of the form (col1,col2,...) creates a multiple-column index. Index values are formed by concatenating the values of the given columns.

Indexes can be created that use only the leading part of column values, using *col_name*(*length*) syntax to specify an index prefix length:

- Prefixes can be specified for CHAR, VARCHAR, BINARY, and VARBINARY columns.
- BLOB and TEXT columns also can be indexed, but a prefix length *must* be given.
- Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for CHAR, VARCHAR, and TEXT columns, and the first *length* bytes of each column value for BINARY, VARBINARY, and BLOB columns.
- For spatial columns, prefix values can be given as described later in this section.

The statement shown here creates an index using the first 10 characters of the name column:

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire name column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up INSERT operations.

Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 1000 bytes long for MyISAM tables, and 767 bytes for InnoDB tables. The NDBCLUSTER storage engine does not support prefixes (see Section 15.1.4.6, “Unsupported or Missing Features in MySQL Cluster”).

Prior to MySQL 4.1.2, the limit is 255 bytes for all storage engines supporting prefixes.

Note

Prefix limits are measured in bytes, whereas the prefix length in CREATE INDEX statements is interpreted as number of characters for nonbinary data types (CHAR, VARCHAR, TEXT). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. This constraint does not apply to NULL values except for the BDB storage engine. For other engines, a UNIQUE index permits multiple NULL values for columns that can contain NULL.

FULLTEXT indexes are supported only for MyISAM tables and can include only CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 11.9, “Full-Text Search Functions”, for details of operation. FULLTEXT indexes are available in MySQL 3.23.23 or later.

The MyISAM storage engine supports spatial columns such as (POINT and GEOMETRY. (Chapter 16, *Spatial Extensions*, describes the spatial data types.) Spatial and nonspatial indexes are available according to the following rules.

Characteristics of spatial indexes (created using SPATIAL INDEX):

- Available only for MyISAM tables in MySQL 4.1 or later.
- Indexed columns must be NOT NULL.
- The full width of each column is indexed by default, but column prefix lengths are permitted. However, as of MySQL 5.0.40, the length is not displayed in SHOW CREATE TABLE output. mysqldump uses that statement. As of that version, if a table with SPATIAL indexes containing prefixed columns is dumped and reloaded, the index is created with no prefixes. (The full column width of each column is indexed.)

Characteristics of nonspatial indexes (created with INDEX, UNIQUE, or PRIMARY KEY):

- Permitted for MyISAM tables.
- Columns can be NULL unless the index is a primary key.
- For each spatial column in a non-SPATIAL index except POINT columns, a column prefix length must be specified. (This is the same requirement as for indexed BLOB columns.) The prefix length is given in bytes.
- The index type for a non-SPATIAL index depends on the storage engine. Currently, B-tree is used.

You can add an index on a column that can have NULL values only if you are using MySQL 3.23.2 or newer and are using the MyISAM, InnoDB, or BDB storage engine. This is also true for MEMORY tables as of MySQL 4.0.2. You can only add an index on a BLOB or TEXT column if you are using MySQL 3.23.2 or newer and are using the MyISAM or BDB storage engine, or MySQL 4.0.14 or newer and the InnoDB storage engine.

An *index_col_name* specification can end with ASC or DESC. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

From MySQL 4.1.0 on, some storage engines permit you to specify an index type when creating an index. The permissible index type values supported by different storage engines are shown in the following table. Where multiple index types are listed, the first one is the default when no index type specifier is given.

Storage Engine	Permissible Index Types
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE
NDB (MySQL 4.1.3 and later)	HASH, BTREE (see note in text)

Example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index USING BTREE ON lookup (id);
```

Note

BTREE indexes are implemented by the NDBCLUSTER storage engine as T-tree indexes.

For indexes on NDBCLUSTER table columns, the USING clause can be specified only for a unique index or primary key. In such cases, the USING HASH clause prevents the creation of an implicit ordered index. Without USING HASH, a statement defining a unique index or primary key automatically results in the creation of a HASH index in addition to the ordered index, both of which index the same set of columns.

The *index_type* clause cannot be used together with SPATIAL INDEX.

If you specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type. The parser recognizes RTREE as a type name, but currently this cannot be specfied for any storage engine.

TYPE *type_name* is recognized as a synonym for USING *type_name*. However, USING is the preferred form.

12.1.5 CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    select_statement
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_tbl_name | (LIKE old_tbl_name) }
```

```
create_definition:
    col_name column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  | {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
      [index_name] [index_type] (index_col_name,...)
  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (index_col_name,...) reference_definition
  | CHECK (expr)

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string'] [reference_definition]

data_type:
    TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME
  | TIMESTAMP
  | DATETIME
  | YEAR
  | CHAR[(length)]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | VARCHAR(length)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | BINARY[(length)]
  | VARBINARY(length)
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | TEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | MEDIUMTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | LONGTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | ENUM(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | SET(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | spatial_type

index_col_name:
    col_name [(length)] [ASC | DESC]
```

```
index_type:
    USING {BTREE | HASH}

reference_definition:
    REFERENCES tbl_name (index_col_name,...)
      [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION

table_options:
    table_option [[,] table_option] ...

table_option:
    {ENGINE|TYPE} = engine_name
  | AUTO_INCREMENT = value
  | AVG_ROW_LENGTH = value
  | [DEFAULT] CHARACTER SET = charset_name
  | CHECKSUM = {0 | 1}
  | [DEFAULT] COLLATE = collation_name
  | COMMENT = 'string'
  | DATA DIRECTORY = 'absolute path to directory'
  | DELAY_KEY_WRITE = {0 | 1}
  | INDEX DIRECTORY = 'absolute path to directory'
  | INSERT_METHOD = { NO | FIRST | LAST }
  | MAX_ROWS = value
  | MIN_ROWS = value
  | PACK_KEYS = {0 | 1 | DEFAULT}
  | PASSWORD = 'string'
  | RAID_TYPE = { 1 | STRIPED | RAID0 }
      RAID_CHUNKS = value
      RAID_CHUNKSIZE = value
  | ROW_FORMAT = {DEFAULT|DYNAMIC|FIXED|COMPRESSED}
  | UNION = (tbl_name[,tbl_name]...)

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some legal select statement)
```

`CREATE TABLE` creates a table with the given name. You must have the `CREATE` [491] privilege for the table.

Rules for permissible table names are given in Section 8.2, “Database, Table, Index, Column, and Alias Names”. By default, the table is created in the default database. An error occurs if the table exists, if there is no default database, or if the database does not exist.

In MySQL 3.22 or later, the table name can be specified as *`db_name.tbl_name`* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write `` `mydb`.`mytbl` ``, not `` `mydb.mytbl` ``.

From MySQL 3.23 on, you can use the `TEMPORARY` keyword when creating a table. A `TEMPORARY` table is visible only to the current connection, and is dropped automatically when the connection is closed. This means that two different connections can use the same temporary table name without conflicting with each other or with an existing non-`TEMPORARY` table of the same name. (The existing table is hidden until the temporary table is dropped.) From MySQL 4.0.2 on, to create temporary tables, you must have the `CREATE TEMPORARY TABLES` [491] privilege.

Note

`CREATE TABLE` does not automatically commit the current active transaction if you use the `TEMPORARY` keyword.

In MySQL 3.23 or later, the keywords IF NOT EXISTS prevent an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the CREATE TABLE statement.

MySQL represents each table by an .frm table format (definition) file in the database directory. The storage engine for the table might create other files as well. In the case of MyISAM tables, the storage engine creates data and index files. Thus, for each MyISAM table *tbl_name*, there are three disk files.

File	Purpose
tbl_name.frm	Table format (definition) file
tbl_name.MYD	Data file
tbl_name.MYI	Index file

Chapter 13, *Storage Engines*, describes what files each storage engine creates to represent tables.

data_type represents the data type in a column definition. *spatial_type* represents a spatial data type. The data type syntax shown is representative only. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type, see Chapter 10, *Data Types*, and Chapter 16, *Spatial Extensions*.

Some attributes do not apply to all data types. AUTO_INCREMENT applies only to integer and floating-point types. DEFAULT does not apply to the BLOB or TEXT types.

- If neither NULL nor NOT NULL is specified, the column is treated as though NULL had been specified.
- An integer or floating-point column can have the additional attribute AUTO_INCREMENT. When you insert a value of NULL (recommended) or 0 into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically this is *value*+1, where *value* is the largest value for the column currently in the table. AUTO_INCREMENT sequences begin with 1.

  To retrieve an AUTO_INCREMENT value after inserting a row, use the LAST_INSERT_ID() [874] SQL function or the mysql_insert_id() C API function. See Section 11.13, “Information Functions”, and Section 17.6.6.35, “mysql_insert_id()”.

  As of MySQL 4.1.1, if the NO_AUTO_VALUE_ON_ZERO [458] SQL mode is enabled, you can store 0 in AUTO_INCREMENT columns as 0 without generating a new sequence value. See Section 5.1.6, “Server SQL Modes”.

  Note

  There can be only one AUTO_INCREMENT column per table, it must be indexed, and it cannot have a DEFAULT value. As of MySQL 3.23, an AUTO_INCREMENT column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an AUTO_INCREMENT column that contains 0.

  For MyISAM and BDB tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. See Section 3.6.9, “Using AUTO_INCREMENT”.

  To make MySQL compatible with some ODBC applications, you can find the AUTO_INCREMENT value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

For information about InnoDB and AUTO_INCREMENT, see Section 13.2.5.3, “AUTO_INCREMENT Handling in InnoDB”.

- As of MySQL 4.1, character data types (CHAR, VARCHAR, TEXT) can include CHARACTER SET and COLLATE attributes to specify the character set and collation for the column. For details, see Section 9.1, “Character Set Support”. CHARSET is a synonym for CHARACTER SET. Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

  Also as of 4.1, MySQL interprets length specifications in character column definitions in characters. (Earlier versions interpret them in bytes.) Lengths for BINARY and VARBINARY are in bytes.

- NULL values are handled differently for TIMESTAMP columns than for other column types. Before MySQL 4.1.6, you cannot store a literal NULL in a TIMESTAMP column; setting the column to NULL sets it to the current date and time. Because TIMESTAMP columns behave this way, the NULL and NOT NULL attributes do not apply in the normal way and are ignored if you specify them. On the other hand, to make it easier for MySQL clients to use TIMESTAMP columns, the server reports that such columns can be assigned NULL values (which is true), even though TIMESTAMP never actually contains a NULL value. You can see this when you use DESCRIBE *tbl_name* to get a description of your table.

  Note that setting a TIMESTAMP column to 0 is not the same as setting it to NULL, because 0 is a valid TIMESTAMP value.

- The DEFAULT clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as NOW() [837] or CURRENT_DATE [829]. The exception is that you can specify CURRENT_TIMESTAMP [829] as the default for a TIMESTAMP column as of MySQL 4.1.2. See Section 10.3.1.2, “TIMESTAMP Properties as of MySQL 4.1”.

  If a column definition includes no explicit DEFAULT value, MySQL determines the default value as described in Section 10.1.4, “Data Type Default Values”.

  BLOB and TEXT columns cannot be assigned a default value.

- A comment for a column can be specified with the COMMENT option. The comment is displayed by the SHOW CREATE TABLE and SHOW FULL COLUMNS statements. This option is operational as of MySQL 4.1. (It is permitted but ignored in earlier versions.)

- KEY is normally a synonym for INDEX. From MySQL 4.1, the key attribute PRIMARY KEY can also be specified as just KEY when given in a column definition. This was implemented for compatibility with other database systems.

- A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. This constraint does not apply to NULL values except for the BDB storage engine. For other engines, a UNIQUE index permits multiple NULL values for columns that can contain NULL.

- A PRIMARY KEY is a unique index where all key columns must be defined as NOT NULL. If they are not explicitly declared as NOT NULL, MySQL declares them so implicitly (and silently). A table can have only one PRIMARY KEY. If you do not have a PRIMARY KEY and an application asks for the PRIMARY KEY in your tables, MySQL returns the first UNIQUE index that has no NULL columns as the PRIMARY KEY.

  In InnoDB tables, having a long PRIMARY KEY wastes a lot of space. (See Section 13.2.11, “InnoDB Table and Index Structures”.)

- In the created table, a PRIMARY KEY is placed first, followed by all UNIQUE indexes, and then the nonunique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated UNIQUE keys.

- A PRIMARY KEY can be a multiple-column index. However, you cannot create a multiple-column index using the PRIMARY KEY key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate PRIMARY KEY(*index_col_name, ...*) clause.

- If a PRIMARY KEY or UNIQUE index consists of only one column that has an integer type, you can also refer to the column as _rowid in SELECT statements (new in MySQL 3.23.11).

- In MySQL, the name of a PRIMARY KEY is PRIMARY. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (_2, _3, ...) to make it unique. You can see index names for a table using SHOW INDEX FROM *tbl_name*. See Section 12.4.5.13, "SHOW INDEX Syntax".

- From MySQL 4.1.0 on, some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING *type_name*.

  Example:

  ```
  CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
  ```

  For details about USING, see Section 12.1.4, "CREATE INDEX Syntax".

  For more information about indexes, see Section 7.4.3, "How MySQL Uses Indexes".

- Only the MyISAM, InnoDB, BDB, and (as of MySQL 4.0.2) MEMORY storage engines support indexes on columns that can have NULL values. In other cases, you must declare indexed columns as NOT NULL or an error results.

- For CHAR, VARCHAR, BINARY, and VARBINARY columns, indexes can be created that use only the leading part of column values, using *col_name*(*length*) syntax to specify an index prefix length. BLOB and TEXT columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for CHAR, VARCHAR, and TEXT columns, and the first *length* bytes of each column value for BINARY, VARBINARY, and BLOB columns. Indexing only a prefix of column values like this can make the index file much smaller. See Section 7.4.1, "Column Indexes".

  Only the MyISAM and (as of MySQL 4.0.14) InnoDB storage engines support indexing on BLOB and TEXT columns. For example:

  ```
  CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
  ```

  Prefixes can be up to 1000 bytes long (767 bytes for InnoDB tables). (Before MySQL 4.1.2, the limit is 255 bytes for all tables.) Note that prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE statements is interpreted as number of characters for nonbinary data types (CHAR, VARCHAR, TEXT). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

- An *index_col_name* specification can end with ASC or DESC. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

- When you use ORDER BY or GROUP BY on a TEXT or BLOB column in a SELECT, the server sorts values using only the initial number of bytes indicated by the max_sort_length [420] system variable. See Section 10.4.3, “The BLOB and TEXT Types”.

- In MySQL 3.23.23 or later, you can create special FULLTEXT indexes, which are used for full-text searches. Only the MyISAM table type supports FULLTEXT indexes. They can be created only from CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 11.9, “Full-Text Search Functions”, for details of operation.

- In MySQL 4.1 or later, you can create SPATIAL indexes on spatial data types. Spatial types are supported only for MyISAM tables and indexed columns must be declared as NOT NULL. See Chapter 16, *Spatial Extensions*.

- In MySQL 3.23.44 or later, InnoDB tables support checking of foreign key constraints. See Section 13.2, “The InnoDB Storage Engine”. Note that the FOREIGN KEY syntax in InnoDB is more restrictive than the syntax presented for the CREATE TABLE statement at the beginning of this section: The columns of the referenced table must always be explicitly named. InnoDB supports both ON DELETE and ON UPDATE actions on foreign keys as of MySQL 3.23.50 and 4.0.8, respectively. For the precise syntax, see Section 13.2.5.4, “FOREIGN KEY Constraints”.

  For other storage engines, MySQL Server parses and ignores the FOREIGN KEY and REFERENCES syntax in CREATE TABLE statements. The CHECK clause is parsed but ignored by all storage engines. See Section 1.9.5.6, “Foreign Keys”.

Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including InnoDB, recognizes or enforces the MATCH clause used in referential integrity constraint definitions. Use of an explicit MATCH clause will not have the specified effect, and also causes ON DELETE and ON UPDATE clauses to be ignored. For these reasons, specifying MATCH should be avoided.

The MATCH clause in the SQL standard controls how NULL values in a composite (multiple-column) foreign key are handled when comparing to a primary key. Starting from MySQL 3.23.50, InnoDB does not check foreign key constraints on those foreign key or referenced key values that contain a NULL column. InnoDB essentially implements the semantics defined by MATCH SIMPLE, which permit a foreign key to be all or partially NULL. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table.

Additionally, MySQL and InnoDB require that the referenced columns be indexed for performance. However, the system does not enforce a requirement that the referenced columns be UNIQUE or be declared NOT NULL. The handling of foreign key references to nonunique keys or keys that contain NULL values is not well defined for operations such as UPDATE or DELETE CASCADE. You are advised to use foreign keys that reference only UNIQUE and NOT NULL keys.

Furthermore, InnoDB does not recognize or support “inline REFERENCES specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. InnoDB accepts REFERENCES clauses only when specified as part of a separate FOREIGN KEY specification. For other storage engines, MySQL Server parses and ignores foreign key specifications.

- There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table and depends on the factors discussed in Section D.3.2, “The Maximum Number of Columns Per Table”.

The *table_option* part of the CREATE TABLE syntax can be used in MySQL 3.23 and above. The = that separates an option name and its value is optional as of MySQL 4.1.

The ENGINE and TYPE options specify the storage engine for the table. ENGINE was added in MySQL 4.0.18 (for 4.0) and 4.1.2 (for 4.1). It is the preferred option name as of those versions, and TYPE has become deprecated. TYPE is supported throughout the 4.x series, but likely will be removed in the future.

The ENGINE and TYPE table options take the storage engine names shown in the following table.

Storage Engine	Description
ARCHIVE	The archiving storage engine. See Section 13.7, “The ARCHIVE Storage Engine”.
BDB	Transaction-safe tables with page locking. Also known as BerkeleyDB. See Section 13.5, “The BDB (BerkeleyDB) Storage Engine”.
CSV	Tables that store rows in comma-separated values format. See Section 13.8, “The CSV Storage Engine”.
EXAMPLE	An example engine. See Section 13.6, “The EXAMPLE Storage Engine”.
HEAP	The data for this table is stored only in memory. See Section 13.4, “The MEMORY (HEAP) Storage Engine”.
ISAM	The original MySQL storage engine. See Section 13.10, “The ISAM Storage Engine”.
InnoDB	Transaction-safe tables with row locking and foreign keys. See Section 13.2, “The InnoDB Storage Engine”.
MEMORY	An alias for HEAP. (Actually, as of MySQL 4.1, MEMORY is the preferred term.)
MERGE	A collection of MyISAM tables used as one table. Also known as MRG_MyISAM. See Section 13.3, “The MERGE Storage Engine”.
MyISAM	The binary portable storage engine that is the improved replacement for ISAM. See Section 13.1, “The MyISAM Storage Engine”.
NDBCLUSTER	Clustered, fault-tolerant, memory-based tables. Also known as NDB. See Chapter 15, *MySQL Cluster*.

If a storage engine is specified that is not available, MySQL uses the default engine instead. Normally, this is MyISAM. For example, if a table definition includes the ENGINE=BDB option but the MySQL server does not support BDB tables, the table is created as a MyISAM table. This makes it possible to have a replication setup where you have transactional tables on the master but tables created on the slave are nontransactional (to get more speed). In MySQL 4.1.1, a warning occurs if the storage engine specification is not honored.

The other table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use ALTER TABLE to convert the table to use a different storage engine.

- AUTO_INCREMENT

  The initial AUTO_INCREMENT value for the table. This works for MyISAM only, for MEMORY as of MySQL 4.1.0, and for InnoDB as of MySQL 4.1.2. To set the first auto-increment value for engines that do not support the AUTO_INCREMENT table option, insert a “dummy” row with a value one less than the desired value after creating the table, and then delete the dummy row.

For engines that support the AUTO_INCREMENT table option in CREATE TABLE statements, you can also use ALTER TABLE *tbl_name* AUTO_INCREMENT = *N* to reset the AUTO_INCREMENT value. The value cannot be set lower than the maximum value currently in the column.

- AVG_ROW_LENGTH

  An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

  When you create a MyISAM table, MySQL uses the product of the MAX_ROWS and AVG_ROW_LENGTH options to decide how big the resulting table is. If you do not specify either option, the maximum size for MyISAM data and index files is 4GB. (If your operating system does not support files that large, table sizes are constrained by the operating system limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you do not really need big files, you can decrease the default pointer size by setting the myisam_data_pointer_size [421] system variable, which was added in MySQL 4.1.2. (See Section 5.1.3, “Server System Variables”.) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you may increase the default pointer size by setting this variable. Setting the value to 7 permits table sizes up to 65,536TB.

- [DEFAULT] CHARACTER SET

  Specify a default character set for the table. CHARSET is a synonym for CHARACTER SET. If the character set name is DEFAULT, the database character set is used.

- CHECKSUM

  Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The CHECKSUM TABLE statement reports the checksum. (MyISAM only.)

- [DEFAULT] COLLATE

  Specify a default collation for the table.

- COMMENT

  A comment for the table, up to 60 characters long.

- DATA DIRECTORY, INDEX DIRECTORY

  By using DATA DIRECTORY='*directory*' or INDEX DIRECTORY='*directory*' you can specify where the MyISAM storage engine should put a table's data file and index file. The directory must be the full path name to the directory, not a relative path.

  These options work only for MyISAM tables from MySQL 4.0 on, when you are not using the --skip-symbolic-links [393] option. Your operating system must also have a working, thread-safe realpath() call. See Section 7.10.2, “Using Symbolic Links for Tables on Unix”, for more complete information.

  Important

  Beginning with MySQL 4.1.24, you cannot use path names that contain the MySQL data directory with DATA DIRECTORY or INDEX DIRECTORY. (See Bug #32167.)

- DELAY_KEY_WRITE

  Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the delay_key_write [410] system variable in Section 5.1.3, “Server System Variables”. (MyISAM only.)

- INSERT_METHOD

  If you want to insert data into a MERGE table, you must specify with INSERT_METHOD the table into which the row should be inserted. INSERT_METHOD is an option useful for MERGE tables only. Use a value of FIRST or LAST to have inserts go to the first or last table, or a value of NO to prevent inserts. This option was introduced in MySQL 4.0.0. See Section 13.3, “The MERGE Storage Engine”.

- MAX_ROWS

  The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

  The NDB storage engine treats this value as a maxmimum. If you plan to create very large MySQL Cluster tables (containing millions of rows), you should use this option to insure that NDB allocates sufficient number of index slots in the hash table used for storing hashes of the table's primary keys by setting MAX_ROWS = 2 * *rows*, where *rows* is the number of rows that you expect to insert into the table.

  The maximum MAX_ROWS value is 4294967295; larger values are truncated to this limit.

- MIN_ROWS

  The minimum number of rows you plan to store in the table. The MEMORY storage engine uses this option as a hint about memory use.

- PACK_KEYS

  PACK_KEYS takes effect only with MyISAM tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to DEFAULT tells the storage engine to pack only long CHAR, VARCHAR, BINARY, or VARBINARY columns.

  If you do not use PACK_KEYS, the default is to pack strings, but not numbers. If you use PACK_KEYS=1, numbers are packed as well.

  When packing binary number keys, MySQL uses prefix compression:

  - Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
  - The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

  This means that if you have many equal keys on two consecutive rows, all following “same” keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes storage_size_for_key + pointer_size (where the pointer size is usually 4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have NULL values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is NULL.)

- PASSWORD

This option is unused. If you have a need to scramble your `.frm` files and make them unusable to any other MySQL server, please contact our sales department.

- `RAID_TYPE`

  The `RAID_TYPE` option can help you to exceed the 2GB/4GB limit for the `MyISAM` data file (not the index file) on operating systems that do not support big files. This option is unnecessary and not recommended for file systems that support big files.

  You can get more speed from the I/O bottleneck by putting `RAID` directories on different physical disks. The only permitted `RAID_TYPE` is `STRIPED`. `1` and `RAID0` are aliases for `STRIPED`.

  If you specify the `RAID_TYPE` option for a `MyISAM` table, specify the `RAID_CHUNKS` and `RAID_CHUNKSIZE` options as well. The maximum `RAID_CHUNKS` value is 255. `MyISAM` creates `RAID_CHUNKS` subdirectories named `00`, `01`, `02`, ... `09`, `0a`, `0b`, ... in the database directory. In each of these directories, `MyISAM` creates a file *`tbl_name`*`.MYD`. When writing data to the data file, the `RAID` handler maps the first `RAID_CHUNKSIZE*1024` bytes to the first file, the next `RAID_CHUNKSIZE*1024` bytes to the next file, and so on.

  `RAID_TYPE` works on any operating system, as long as you have built MySQL with the `--with-raid` option to `configure`. To determine whether a server supports `RAID` tables, use `SHOW VARIABLES LIKE 'have_raid'` to see whether the variable value is `YES`.

- `ROW_FORMAT`

  Defines how the rows should be stored. Currently, this option works only with `MyISAM` tables. The option value can be `FIXED` or `DYNAMIC` for static or variable-length row format. `myisampack` sets the type to `COMPRESSED`. See Section 13.1.3, “`MyISAM` Table Storage Formats”.

  Note

  When executing a `CREATE TABLE` statement, if you specify a row format which is not supported by the storage engine that is used for the table, the table is created using that storage engine's default row format. The information reported in this column in response to `SHOW TABLE STATUS` is the actual row format used. This may differ from the value in the `Create_options` column because the original `CREATE TABLE` definition is retained during creation.

- `UNION`

  `UNION` is used when you want to access a collection of identical `MyISAM` tables as one. This works only with `MERGE` tables. See Section 13.3, “The `MERGE` Storage Engine”.

  In MySQL 4.1, you must have `SELECT` [492], `UPDATE` [493], and `DELETE` [491] privileges for the tables you map to a `MERGE` table.

  Note

  Originally, all tables used had to be in the same database as the `MERGE` table itself. This restriction has been lifted as of MySQL 4.1.1.

  Important

  The original `CREATE TABLE` statement, including all specifications and table options are stored by MySQL when the table is created. The information is retained so that if you change storage engines, collations or other settings using an `ALTER`

> TABLE statement, the original table options specified are retained. This enables you to change between InnoDB and MyISAM table types even though the row formats supported by the two engines are different.
>
> Because the text of the original statement is retained, but due to the way that certain values and options may be silently reconfigured (such as the ROW_FORMAT), the active table definition (accessible through DESCRIBE or with SHOW TABLE STATUS) and the table creation string (accessible through SHOW CREATE TABLE) will report different values.

As of MySQL 3.23, you can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

For more information, see Section 12.1.5.1, “CREATE TABLE ... SELECT Syntax”.

In MySQL 4.1, you can also use LIKE to create an empty table based on the definition of another table, including any column attributes and indexes the original table has:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

The copy is created using the same version of the table storage format as the original table.

CREATE TABLE ... LIKE does not preserve any DATA DIRECTORY or INDEX DIRECTORY table options that were specified for the original table, or any foreign key definitions.

If the original table is a TEMPORARY table, CREATE TABLE ... LIKE does not preserve TEMPORARY. To create a TEMPORARY destination table, use CREATE TEMPORARY TABLE ... LIKE.

12.1.5.1 CREATE TABLE ... SELECT Syntax

As of MySQL 3.23, you can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the SELECT. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        TYPE=MyISAM SELECT b,c FROM test2;
```

This creates a MyISAM table with three columns, a, b, and c. Notice that the columns from the SELECT statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM bar;
+------+---+
| m    | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

For each row in table `foo`, a row is inserted in `bar` with the values from `foo` and default values for the new columns.

In a table resulting from `CREATE TABLE ... SELECT`, columns named only in the `CREATE TABLE` part come first. Columns named in both parts or only in the `SELECT` part come after that. The data type of `SELECT` columns can be overridden by also specifying the column in the `CREATE TABLE` part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

You can precede the `SELECT` by `IGNORE` or `REPLACE` to indicate how to handle rows that duplicate unique key values. With `IGNORE`, new rows that duplicate an existing row on a unique key value are discarded. With `REPLACE`, new rows replace rows that have the same unique key value. If neither `IGNORE` nor `REPLACE` is specified, duplicate unique key values result in an error.

`CREATE TABLE ... SELECT` does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the `SELECT` statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of data types might occur. For example, the `AUTO_INCREMENT` attribute is not preserved, and `VARCHAR` columns can become `CHAR` columns. Retrained attributes are `NULL` (or `NOT NULL`) and, for those columns that have them, `CHARACTER SET`, `COLLATION`, `COMMENT`, and the `DEFAULT` clause.

When creating a table with `CREATE TABLE ... SELECT`, make sure to alias any function calls or expressions in the query. If you do not, the `CREATE` statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

As of MySQL 4.1, you can explicitly specify the data type for a generated column:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

For `CREATE TABLE ... SELECT`, if `IF NOT EXISTS` is given and the destination table already exists, MySQL handles the statement as follows:

- The table definition given in the `CREATE TABLE` part is ignored. No error occurs, even if the definition does not match that of the existing table. MySQL attempts to insert the rows from the `SELECT` part anyway.
- If there is a mismatch between the number of columns in the table and the number of columns produced by the `SELECT` part, the selected values are assigned to the rightmost columns. For example, if the table contains *n* columns and the `SELECT` produces *m* columns, where *m* < *n*, the selected values are

assigned to the *m* rightmost columns in the table. Each of the initial *n* – *m* columns is assigned its default value, either that specified explicitly in the column definition or the implicit column data type default if the definition contains no default. If the SELECT part produces too many columns (*m* > *n*), an error occurs.

The following example illustrates IF NOT EXISTS handling:

```
mysql> CREATE TABLE t1 (i1 INT DEFAULT 0, i2 INT, i3 INT, i4 INT);
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE IF NOT EXISTS t1 (c1 CHAR(10)) SELECT 1, 2;
Query OK, 1 row affected, 1 warning (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1;
+------+------+------+------+
| i1   | i2   | i3   | i4   |
+------+------+------+------+
|    0 | NULL |    1 |    2 |
+------+------+------+------+
1 row in set (0.00 sec)
```

To ensure that the update log or binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts for CREATE TABLE ... SELECT statements.

12.1.5.2 Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a CREATE TABLE or ALTER TABLE statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

Possible data type changes are given in the following list.

- VARCHAR columns with a length less than four are changed to CHAR.
- If any column in a table has a variable length, the entire row becomes variable-length as a result. Therefore, if a table contains any variable-length columns (VARCHAR, TEXT, or BLOB), all CHAR columns longer than three characters are changed to VARCHAR columns. This does not affect how you use the columns in any way; in MySQL, VARCHAR is just a different way to store characters. MySQL performs this conversion because it saves space and makes table operations faster. See Chapter 13, *Storage Engines*.
- From MySQL 4.1.0 onward, a CHAR or VARCHAR column with a length specification greater than 255 is converted to the smallest TEXT type that can hold values of the given length. For example, VARCHAR(500) is converted to TEXT, and VARCHAR(200000) is converted to MEDIUMTEXT. Similar conversions occur for BINARY and VARBINARY, except that they are converted to a BLOB type.

  Note that these conversions result in a change in behavior with regard to treatment of trailing spaces.
- From MySQL 4.1.2 on, specifying the CHARACTER SET binary attribute for a character data type causes the column to be created as the corresponding binary data type: CHAR becomes BINARY, VARCHAR becomes VARBINARY, and TEXT becomes BLOB. For the ENUM and SET data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
```

```
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

- For a specification of DECIMAL(*M*,*D*), if *M* is not larger than *D*, it is adjusted upward. For example, DECIMAL(10,10) becomes DECIMAL(11,10).

Other silent column specification changes include modifications to attribute or index specifications:

- TIMESTAMP display sizes are discarded from MySQL 4.1 on, due to changes made to the TIMESTAMP data type in that version. Before MySQL 4.1, TIMESTAMP display sizes must be even and in the range from 2 to 14. If you specify a display size of 0 or greater than 14, the size is coerced to 14. Odd-valued sizes in the range from 1 to 13 are coerced to the next higher even number.

  Also note that, in MySQL 4.1 and later, TIMESTAMP columns are NOT NULL by default.

- Before MySQL 4.1.6, you cannot store a literal NULL in a TIMESTAMP column; setting it to NULL sets it to the current date and time. Because TIMESTAMP columns behave this way, the NULL and NOT NULL attributes do not apply in the normal way and are ignored if you specify them. DESCRIBE *tbl_name* always reports that a TIMESTAMP column can be assigned NULL values.

- Columns that are part of a PRIMARY KEY are made NOT NULL even if not declared that way.

- Starting from MySQL 3.23.51, trailing spaces are automatically deleted from ENUM and SET member values when the table is created.

- MySQL maps certain data types used by other SQL database vendors to MySQL types. See Section 10.7, “Using Data Types from Other Database Engines”.

- If you include a USING clause to specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.

To see whether MySQL used a data type other than the one you specified, issue a DESCRIBE or SHOW CREATE TABLE statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using myisampack. See Section 13.1.3.3, “Compressed Table Characteristics”.

12.1.6 DROP DATABASE Syntax

```
DROP DATABASE [IF EXISTS] db_name
```

DROP DATABASE drops all tables in the database and deletes the database. Be *very* careful with this statement! To use DROP DATABASE, you need the DROP [491] privilege on the database.

Important

When a database is dropped, user privileges on the database are *not* automatically dropped. See Section 12.4.1.2, “GRANT Syntax”.

In MySQL 3.22 or later, you can use the keywords IF EXISTS to prevent an error from occurring if the database does not exist.

As of MySQL 4.1.1, if the default database is dropped, the default database is unset (the DATABASE() [872] function returns NULL).

If you use DROP DATABASE on a symbolically linked database, both the link and the original database are deleted.

As of MySQL 4.1.2, DROP DATABASE returns the number of tables that were removed. This corresponds to the number of .frm files removed.

The DROP DATABASE statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with the following extensions.

.BAK	.DAT	.HSH	.ISD
.ISM	.MRG	.MYD	.MYI
.db	.frm	.ibd	.ndb

- All subdirectories with names that consist of two hex digits 00-ff. These are subdirectories used for RAID tables. (These directories are not removed in versions of MySQL after 4.1, where support for RAID tables is removed. You should convert any existing RAID tables and remove these directories manually before upgrading to later MySQL versions.)
- The db.opt file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the DROP DATABASE statement again.

You can also drop databases with mysqladmin. See Section 4.5.2, "mysqladmin — Client for Administering a MySQL Server".

12.1.7 DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX drops the index named *index_name* from the table *tbl_name*. In MySQL 3.22 or later, DROP INDEX is mapped to an ALTER TABLE statement to drop the index. See Section 12.1.2, "ALTER TABLE Syntax". DROP INDEX does not do anything prior to MySQL 3.22.

12.1.8 DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

DROP TABLE removes one or more tables. You must have the DROP [491] privilege for each table. All table data and the table definition are *removed*, so *be careful* with this statement! If any of the tables named in the argument list do not exist, MySQL returns an error indicating by name which nonexisting tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Important

When a table is dropped, user privileges on the table are *not* automatically dropped. See Section 12.4.1.2, “GRANT Syntax”.

In MySQL 3.22 or later, you can use the keywords IF EXISTS to prevent an error from occurring for tables that do not exist. As of MySQL 4.1, a NOTE is generated for each nonexistent table when using IF EXISTS. See Section 12.4.5.26, “SHOW WARNINGS Syntax”.

RESTRICT and CASCADE are permitted to make porting easier. In MySQL 4.1 and earlier, they do nothing.

Note

DROP TABLE automatically commits the current active transaction, unless you are using MySQL 4.1 or higher and the TEMPORARY keyword.

The TEMPORARY keyword is ignored in MySQL 4.0. As of 4.1, it has the following effect:

- The statement drops only TEMPORARY tables.
- The statement does not end an ongoing transaction.
- No access rights are checked. (A TEMPORARY table is visible only to the session that created it, so no check is necessary.)

Using TEMPORARY is a good way to ensure that you do not accidentally drop a non-TEMPORARY table.

12.1.9 RENAME TABLE Syntax

```
RENAME TABLE tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

This statement renames one or more tables. It was added in MySQL 3.23.23.

The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running. For example, if you have an existing table old_table, you can create another table new_table that has the same structure but is empty, and then replace the existing table with the empty one as follows (assuming that backup_table does not already exist):

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that tmp_table does not already exist):

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

As long as two databases are on the same file system, you can use RENAME TABLE to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

Any privileges granted specifically for the renamed table or view are not migrated to the new name. They must be changed manually.

When you execute RENAME, you cannot have any locked tables or active transactions. You must also have the ALTER [491] and DROP [491] privileges on the original table, and the CREATE [491] and INSERT [492] privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it does a reverse rename for all renamed tables to return everything to its original state.

You cannot use RENAME to rename a TEMPORARY table. However, you can use ALTER TABLE instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

12.1.10 TRUNCATE TABLE Syntax

```
TRUNCATE [TABLE] tbl_name
```

TRUNCATE TABLE empties a table completely. Logically, this is equivalent to a DELETE statement that deletes all rows, but there are practical differences under some circumstances.

For InnoDB, TRUNCATE TABLE is mapped to DELETE, so there is no difference.

For other storage engines, TRUNCATE TABLE differs from DELETE in the following ways from MySQL 4.0 onward:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- As of MySQL 4.1.13, truncate operations cause an implicit commit. Before 4.1.13, truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction.
- Truncation operations cannot be performed if the session holds an active table lock.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is “0 rows affected,” which should be interpreted as “no information.”
- As long as the table format file *tbl_name*.frm is valid, the table can be re-created as an empty table with TRUNCATE TABLE, even if the data or index files have become corrupted.
- The table handler does not remember the last used AUTO_INCREMENT value, but starts counting from the beginning. This is true even for MyISAM and InnoDB, which normally do not reuse sequence values. (Some older versions may not reset the AUTO_INCREMENT value. In this case, you can use ALTER TABLE *tbl_name* AUTO_INCREMENT=1 after the TRUNCATE TABLE statement.)

In MySQL 3.23, TRUNCATE TABLE is mapped to COMMIT; DELETE FROM *tbl_name*, so it behaves like DELETE. See Section 12.2.1, “DELETE Syntax”.

TRUNCATE TABLE was added in MySQL 3.23.28, although from 3.23.28 to 3.23.32, the keyword TABLE must be omitted.

12.2 Data Manipulation Statements

12.2.1 DELETE Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

For the single-table syntax, the DELETE statement deletes rows from *tbl_name*. The number of rows deleted can be determined by calling the mysql_info() C API function. The WHERE clause, if given, specifies the conditions that identify which rows to delete. With no WHERE clause, all rows are deleted. If the ORDER BY clause is specified, the rows are deleted in the order that is specified. The LIMIT clause places a limit on the number of rows that can be deleted.

For the multiple-table syntax, DELETE deletes from each *tbl_name* the rows that satisfy the conditions. In this case, ORDER BY and LIMIT cannot be used.

where_condition is an expression that evaluates to true for each row to be deleted. It is specified as described in Section 12.2.7, “SELECT Syntax”.

Currently, you cannot delete from a table and select from the same table in a subquery.

You need the DELETE [491] privilege on a table to delete rows from it. You need only the SELECT [492] privilege for any columns that are only read, such as those named in the WHERE clause.

As stated, a DELETE statement with no WHERE clause deletes all rows. A faster way to do this, when you do not need to know the number of deleted rows, is to use TRUNCATE TABLE. However, within a transaction or if you have a lock on the table, TRUNCATE TABLE cannot be used whereas DELETE can. See Section 12.1.10, “TRUNCATE TABLE Syntax”, and Section 12.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”.

In MySQL 3.23, DELETE without a WHERE clause returns zero as the number of affected rows.

In MySQL 3.23, if you really want to know how many rows are deleted when you are deleting all rows, and are willing to suffer a speed penalty, you can use a DELETE statement that includes a WHERE clause with an expression that is true for every row. For example:

```
mysql> DELETE FROM tbl_name WHERE 1>0;
```

This is much slower than TRUNCATE *tbl_name*, because it deletes rows one at a time.

If you delete the row containing the maximum value for an AUTO_INCREMENT column, the value is reused later for an ISAM or BDB table, but not for a MyISAM or InnoDB table. If you delete all rows in the table with DELETE FROM *tbl_name* (without a WHERE clause) in autocommit [406] mode, the sequence starts over for all storage engines except InnoDB and (as of MySQL 4.0) MyISAM. There are some exceptions

to this behavior for InnoDB tables, as discussed in Section 13.2.5.3, “AUTO_INCREMENT Handling in InnoDB”.

For MyISAM and BDB tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for MyISAM tables. See Section 3.6.9, “Using AUTO_INCREMENT”.

The DELETE statement supports the following modifiers:

- If you specify LOW_PRIORITY, the server delays execution of the DELETE until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).
- For MyISAM tables, if you use the QUICK keyword, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The IGNORE keyword causes MySQL to ignore all errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of IGNORE are returned as warnings. This option first appeared in MySQL 4.1.1.

The speed of delete operations may also be affected by factors discussed in Section 7.3.2.3, “Speed of DELETE Statements”.

In MyISAM tables, deleted rows are maintained in a linked list and subsequent INSERT operations reuse old row positions. To reclaim unused space and reduce file sizes, use the OPTIMIZE TABLE statement or the myisamchk utility to reorganize tables. OPTIMIZE TABLE is easier to use, but myisamchk is faster. See Section 12.4.2.5, “OPTIMIZE TABLE Syntax”, and Section 4.6.2, “myisamchk — MyISAM Table-Maintenance Utility”.

The QUICK modifier affects whether index leaves are merged for delete operations. DELETE QUICK is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

DELETE QUICK is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of QUICK can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed AUTO_INCREMENT column.
2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.
3. Delete a block of rows at the low end of the column range using DELETE QUICK.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of QUICK. They remain underfilled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain underfilled even if you later use DELETE without QUICK, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, use OPTIMIZE TABLE.

If you are going to delete many rows from a table, it might be faster to use DELETE QUICK followed by OPTIMIZE TABLE. This rebuilds the index rather than performing many index block merge operations.

The MySQL-specific LIMIT *row_count* option to DELETE tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a given DELETE statement does not take too much time. You can simply repeat the DELETE statement until the number of affected rows is less than the LIMIT value.

If the DELETE statement includes an ORDER BY clause, rows are deleted in the order specified by the clause. This is useful primarily in conjunction with LIMIT. For example, the following statement finds rows matching the WHERE clause, sorts them by timestamp_column, and deletes the first (oldest) one:

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

ORDER BY may also be useful in some cases to delete rows in an order required to avoid referential integrity violations.

ORDER BY can be used with DELETE beginning with MySQL 4.0.0.

From MySQL 4.0, you can specify multiple tables in the DELETE statement to delete rows from one or more tables depending on a particular condition in multiple tables. However, you cannot use ORDER BY or LIMIT in a multiple-table DELETE.

If you are deleting many rows from a large table, you may exceed the lock table size for an InnoDB table. To avoid this problem, or simply to minimize the time that the table remains locked, the following strategy (which does not use DELETE at all) might be helpful:

1. Select the rows *not* to be deleted into an empty table that has the same structure as the original table:

   ```
   INSERT INTO t_copy SELECT * FROM t WHERE ... ;
   ```

2. Use RENAME TABLE to atomically move the original table out of the way and rename the copy to the original name:

   ```
   RENAME TABLE t TO t_old, t_copy TO t;
   ```

3. Drop the original table:

   ```
   DROP TABLE t_old;
   ```

No other sessions can access the tables involved while RENAME TABLE executes, so the rename operation is not subject to concurrency problems. See Section 12.1.9, “RENAME TABLE Syntax”.

You can specify multiple tables in a DELETE statement to delete rows from one or more tables depending on the particular condition in the WHERE clause. However, you cannot use ORDER BY or LIMIT in a multiple-table DELETE. The *table_references* clause lists the tables involved in the join. Its syntax is described in Section 12.2.7.1, “JOIN Syntax”.

The first multiple-table DELETE syntax is supported starting from MySQL 4.0.0. The second is supported starting from MySQL 4.0.2.

For the first multiple-table syntax, only matching rows from the tables listed before the FROM clause are deleted. For the second multiple-table syntax, only matching rows from the tables listed in the FROM clause (before the USING clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
```

```
WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables `t1` and `t2`.

The preceding examples use `INNER JOIN`, but multiple-table `DELETE` statements can use other types of join permitted in `SELECT` statements, such as `LEFT JOIN`. For example, to delete rows that exist in `t1` that have no match in `t2`, use a `LEFT JOIN`:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

The syntax permits `.*` after each *`tbl_name`* for compatibility with `Access`.

If you use a multiple-table `DELETE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the `ON DELETE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly.

Table aliases in a multiple-table `DELETE` should be declared only in the *`table_references`* part of the statement. Elsewhere, aliases references are permitted but should not be declared.

Note

The syntax for multiple-table `DELETE` statements that use table aliases changed between MySQL 4.0 and 4.1. In MySQL 4.0, you should use the true table name to refer to any table from which rows should be deleted:

```
DELETE test FROM test AS t1, test2 WHERE ...
```

In MySQL 4.1, if you declare an alias for a table, you must use the alias when referring to the table:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

We did not make this change in 4.0 to avoid breaking any old 4.0 applications that were using the old syntax. However, if you use such `DELETE` statements and are using replication, the change in syntax means that a 4.0 master cannot replicate to 4.1 (or higher) slaves.

For multiple-table deletes, prior to MySQL 4.1.2 you must refer to the tables without using aliases. For example:

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

As of MySQL 4.1.2, aliases can be used, but for alias references in the list of tables from which to delete rows, the default database is used unless one is specified explicitly. For example, if the default database is `db1`, the following statement does not work because the unqualified alias reference `a2` is interpreted as having a database of `db1`:

```
DELETE a1, a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

To correctly match an alias that refers to a table outside the default database, you must explicitly qualify the reference with the name of the proper database:

```
DELETE a1, db2.a2 FROM db1.t1 AS a1 INNER JOIN db2.t2 AS a2
WHERE a1.id=a2.id;
```

12.2.2 DO Syntax

```
DO expr [, expr] ...
```

DO executes the expressions but does not return any results. In most respects, DO is shorthand for SELECT *expr, ...*, but has the advantage that it is slightly faster when you do not care about the result.

DO is useful primarily with functions that have side effects, such as RELEASE_LOCK() [879].

DO was added in MySQL 3.23.47.

12.2.3 HANDLER Syntax

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

The HANDLER statement provides direct access to table storage engine interfaces. It is available for MyISAM tables as MySQL 4.0.0 and InnoDB tables as of MySQL 4.0.3.

The HANDLER ... OPEN statement opens a table, making it accessible using subsequent HANDLER ... READ statements. This table object is not shared by other sessions and is not closed until the session calls HANDLER ... CLOSE or the session terminates. If you open the table using an alias, further references to the open table with other HANDLER statements must use the alias rather than the table name.

The first HANDLER ... READ syntax fetches a row where the index specified satisfies the given values and the WHERE condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index my_idx includes three columns named col_a, col_b, and col_c, in that order. The HANDLER statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the HANDLER interface to refer to a table's PRIMARY KEY, use the quoted identifier `PRIMARY`:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second HANDLER ... READ syntax fetches a row from the table in index order that matches the WHERE condition.

The third HANDLER ... READ syntax fetches a row from the table in natural row order that matches the WHERE condition. It is faster than HANDLER *tbl_name* READ *index_name* when a full table scan is

desired. Natural row order is the order in which rows are stored in a MyISAM table data file. This statement works for InnoDB tables as well, but there is no such concept because there is no separate data file.

Without a LIMIT clause, all forms of HANDLER ... READ fetch a single row if one is available. To return a specific number of rows, include a LIMIT clause. It has the same syntax as for the SELECT statement. See Section 12.2.7, “SELECT Syntax”.

HANDLER ... CLOSE closes a table that was opened with HANDLER ... OPEN.

There are several reasons to use the HANDLER interface instead of normal SELECT statements:

- HANDLER is faster than SELECT:
  - A designated storage engine handler object is allocated for the HANDLER ... OPEN. The object is reused for subsequent HANDLER statements for that table; it need not be reinitialized for each one.
  - There is less parsing involved.
  - There is no optimizer or query-checking overhead.
  - The table does not have to be locked between two handler requests.
  - The handler interface does not have to provide a consistent look of the data (for example, dirty reads are permitted), so the storage engine can use optimizations that SELECT does not normally permit.
- For applications that use a low-level ISAM-like interface, HANDLER makes it much easier to port them to MySQL.
- HANDLER enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with SELECT. The HANDLER interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

HANDLER is a somewhat low-level statement. For example, it does not provide consistency. That is, HANDLER ... OPEN does *not* take a snapshot of the table, and does *not* lock the table. This means that after a HANDLER ... OPEN statement is issued, table data can be modified (by the current session or other sessions) and these modifications might be only partially visible to HANDLER ... NEXT or HANDLER ... PREV scans.

An open handler can be closed and marked for reopen, in which case the handler loses its position in the table. This occurs when both of the following circumstances are true:

- Any session executes FLUSH TABLES or DDL statements on the handler's table.
- The session in which the handler is open executes non-HANDLER statements that use tables.

12.2.4 INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
```

```
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

`INSERT` inserts new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. The `INSERT ... VALUES` form with multiple value lists is supported in MySQL 3.22.5 or later. The `INSERT ... SET` syntax is supported in MySQL 3.22.10 or later. `INSERT ... SELECT` is discussed further in Section 12.2.4.1, "`INSERT ... SELECT` Syntax".

You can use `REPLACE` instead of `INSERT` to overwrite old rows. `REPLACE` is the counterpart to `INSERT IGNORE` in the treatment of new rows that contain unique key values that duplicate old rows: The new rows are used to replace the old rows rather than being discarded. See Section 12.2.6, "`REPLACE` Syntax".

`tbl_name` is the table into which rows should be inserted. The columns for which the statement provides values can be specified as follows:

- You can provide a comma-separated list of column names following the table name. In this case, a value for each named column must be provided by the `VALUES` list or the `SELECT` statement.
- If you do not specify a list of column names for `INSERT ... VALUES` or `INSERT ... SELECT`, values for every column in the table must be provided by the `VALUES` list or the `SELECT` statement. If you do not know the order of the columns in the table, use `DESCRIBE tbl_name` to find out.
- The `SET` clause indicates the column names explicitly.

Column values can be given in several ways:

- Normally, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 10.1.4, "Data Type Default Values", and Section 1.9.6.2, "Constraints on Invalid Data".
- You can use the keyword `DEFAULT` to explicitly set a column to its default value. (New in MySQL 4.0.3.) This makes it easier to write `INSERT` statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete `VALUES` list that does not include a value for each column in the table. Otherwise, you would have to write out the list of column names corresponding to each value in the `VALUES` list.

  As of MySQL 4.1.0, you can use `DEFAULT(col_name)` [877] as a more general form that can be used in expressions to produce a given column's default value.
- If both the column list and the `VALUES` list are empty, `INSERT` creates a row with each column set to its default value:

```
INSERT INTO tbl_name () VALUES();
```

- You can specify an expression *expr* to provide a column value. This might involve type conversion if the type of the expression does not match the type of the column, and conversion of a given value can result in different inserted values depending on the data type. For example, inserting the string `'1999.0e-2'` into an `INT`, `FLOAT`, `DECIMAL(10,6)`, or `YEAR` column results in the values `1999`, `19.9921`, `19.992100`, and `1999` being inserted, respectively. The reason the value stored in the `INT` and `YEAR` columns is `1999` is that the string-to-integer conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the floating-point and fixed-point columns, the string-to-floating-point conversion considers the entire string a valid floating-point value.

  An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for `col2` refers to `col1`, which has previously been assigned:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  But the following is not legal, because the value for `col1` refers to `col2`, which is assigned after `col1`:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

  One exception involves columns that contain `AUTO_INCREMENT` values. Because the `AUTO_INCREMENT` value is generated after other value assignments, any reference to an `AUTO_INCREMENT` column in the assignment returns a `0`.

`INSERT` statements that use `VALUES` syntax can insert multiple rows. To do this, include multiple lists of column values, each enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

The values list for each row must be enclosed within parentheses. The following statement is illegal because the number of values in the list does not match the number of column names:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

`VALUE` is a synonym for `VALUES` in this context. Neither implies anything about the number of values lists, and either may be used whether there is a single values list or multiple lists.

The affected-rows value for an `INSERT` can be obtained using the `mysql_affected_rows()` C API function (see Section 17.6.6.1, “`mysql_affected_rows()`”).

If you use an `INSERT ... VALUES` statement with multiple value lists or `INSERT ... SELECT`, the statement returns an information string in this format:

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Records` indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because `Duplicates` can be nonzero.) `Duplicates` indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT INTO ... SELECT` statements, the column is set to the implicit default value for the column data type. This is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. `INSERT INTO ... SELECT` statements are handled the same way as multiple-row inserts because the server does not examine the result set from the `SELECT` to see whether it returns

a single row. (For a single-row INSERT, no warning occurs when NULL is inserted into a NOT NULL column. Instead, the statement fails with an error.)

- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.
- Assigning a value such as '10.34 a' to a numeric column. The trailing nonnumeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to 0.
- Inserting a string into a string column (CHAR, VARCHAR, TEXT, or BLOB) that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

If you are using the C API, the information string can be obtained by invoking the mysql_info() function. See Section 17.6.6.33, "mysql_info()".

If INSERT inserts a row into a table that has an AUTO_INCREMENT column, you can find the value used for that column by using the SQL LAST_INSERT_ID() [874] function. From within the C API, use the mysql_insert_id() function. However, you should note that the two functions do not always behave identically. The behavior of INSERT statements with respect to AUTO_INCREMENT columns is discussed further in Section 11.13, "Information Functions", and Section 17.6.6.35, "mysql_insert_id()".

The INSERT statement supports the following modifiers:

- If you use the DELAYED keyword, the server puts the row or rows to be inserted into a buffer, and the client issuing the INSERT DELAYED statement can then continue immediately. If the table is in use, the server holds the rows. When the table is free, the server begins inserting rows, checking periodically to see whether there are any new read requests for the table. If there are, the delayed row queue is suspended until the table becomes free again. See Section 12.2.4.2, "INSERT DELAYED Syntax". DELAYED was added in MySQL 3.22.5.

  DELAYED is ignored with INSERT ... SELECT or INSERT ... ON DUPLICATE KEY UPDATE.

  If you use the LOW_PRIORITY keyword, execution of the INSERT is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the INSERT LOW_PRIORITY statement is waiting. It is possible, therefore, for a client that issues an INSERT LOW_PRIORITY statement to wait for a very long time (or even forever) in a read-heavy environment. (This is in contrast to INSERT DELAYED, which lets the client continue at once.) Note that LOW_PRIORITY should normally not be used with MyISAM tables because doing so disables concurrent inserts. See Section 7.6.3, "Concurrent Inserts". LOW_PRIORITY was added in MySQL 3.22.5.

  LOW_PRIORITY and HIGH_PRIORITY affect only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

- If you specify HIGH_PRIORITY, it overrides the effect of the --low-priority-updates [389] option if the server was started with that option. It also causes concurrent inserts not to be used. See Section 7.6.3, "Concurrent Inserts". HIGH_PRIORITY was added in MySQL 3.23.11.
- If you use the IGNORE keyword, errors that occur while executing the INSERT statement are treated as warnings instead. For example, without IGNORE, a row that duplicates an existing UNIQUE index or PRIMARY KEY value in the table causes a duplicate-key error and the statement is aborted. With IGNORE, the row still is not inserted, but no error is issued. Data conversions that would trigger errors

abort the statement if IGNORE is not specified. With IGNORE, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the mysql_info() C API function how many rows were actually inserted into the table.

- If you specify ON DUPLICATE KEY UPDATE, and a row is inserted that would cause a duplicate value in a UNIQUE index or PRIMARY KEY, an UPDATE of the old row is performed. is performed. The affected-rows value per row is 1 if the row is inserted as a new row and 2 if an existing row is updated. See Section 12.2.4.3, "INSERT ... ON DUPLICATE KEY UPDATE Syntax". ON DUPLICATE KEY UPDATE was added in MySQL 4.1.0.

Inserting into a table requires the INSERT [492] privilege for the table. If the ON DUPLICATE KEY UPDATE clause is used and a duplicate key causes an UPDATE to be performed instead, the statement requires the UPDATE [493] privilege for the columns to be updated. For columns that are read but not modified you need only the SELECT [492] privilege (such as for a column referenced only on the right hand side of an *col_name*=*expr* assignment in an ON DUPLICATE KEY UPDATE clause).

12.2.4.1 INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

With INSERT ... SELECT, you can quickly insert many rows into a table from one or many tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for a INSERT ... SELECT statements:

- Prior to MySQL 4.0.1, INSERT ... SELECT implicitly operates in IGNORE mode. As of MySQL 4.0.1, specify IGNORE explicitly to ignore rows that would cause duplicate-key violations.
- DELAYED is ignored with INSERT ... SELECT.
- Prior to MySQL 4.0.14, the target table of the INSERT statement cannot appear in the FROM clause of the SELECT part of the query. This limitation is lifted in 4.0.14. In this case, MySQL creates a temporary table to hold the rows from the SELECT and then inserts those rows into the target table. However, it remains true that you cannot use INSERT INTO t ... SELECT ... FROM t when t is a TEMPORARY table, because TEMPORARY tables cannot be referred to twice in the same statement (see Section B.5.7.2, "TEMPORARY Table Problems").
- AUTO_INCREMENT columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts for INSERT ... SELECT statements.
- Currently, you cannot insert into a table and select from the same table in a subquery.
- To avoid ambiguous column reference problems when the SELECT and the INSERT refer to the same table, provide a unique alias for each table used in the SELECT part, and qualify column names in that part with the appropriate alias.

In the values part of ON DUPLICATE KEY UPDATE, you can refer to columns in other tables, as long as you do not use GROUP BY in the SELECT part. One side effect is that you must qualify nonunique column names in the values part.

12.2.4.2 INSERT DELAYED Syntax

```
INSERT DELAYED ...
```

The DELAYED option for the INSERT statement is a MySQL extension to standard SQL that is very useful if you have clients that cannot or need not wait for the INSERT to complete. This is a common situation when you use MySQL for logging and you also periodically run SELECT and UPDATE statements that take a long time to complete. DELAYED was introduced in MySQL 3.22.15.

When a client uses INSERT DELAYED, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using INSERT DELAYED is that inserts from many clients are bundled together and written in one block. This is much faster than performing many separate inserts.

Note that INSERT DELAYED is slower than a normal INSERT if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are delayed rows. This means that you should use INSERT DELAYED only when you are really sure that you need it.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate mysqld forcibly (for example, with kill -9) or if mysqld dies unexpectedly, *any queued rows that have not been written to disk are lost*.

There are some constraints on the use of DELAYED:

- INSERT DELAYED works only with ISAM, MyISAM, and (beginning with MySQL 4.1) MEMORY tables. For engines that do not support DELAYED, an error occurs.
- An error occurs for INSERT DELAYED if used with a table that has been locked with LOCK TABLES because the insert must be handled by a separate thread, not by the session that holds the lock.
- For MyISAM tables, if there are no free blocks in the middle of the data file, concurrent SELECT and INSERT statements are supported. Under these circumstances, you very seldom need to use INSERT DELAYED with MyISAM.
- INSERT DELAYED should be used only for INSERT statements that specify value lists. This is enforced as of MySQL 4.0.18. The server ignores DELAYED for INSERT ... SELECT or INSERT ... ON DUPLICATE KEY UPDATE statements.
- Because the INSERT DELAYED statement returns immediately, before the rows are inserted, you cannot use LAST_INSERT_ID() [874] to get the AUTO_INCREMENT value that the statement might generate.
- DELAYED rows are not visible to SELECT statements until they actually have been inserted.
- INSERT DELAYED is treated as a normal INSERT if the statement inserts multiple rows and binary logging is enabled.
- DELAYED is ignored on slave replication servers, so that INSERT DELAYED is treated as a normal INSERT on slaves. This is because DELAYED could cause the slave to have different data than the master.
- Pending INSERT DELAYED statements are lost if a table is write locked and ALTER TABLE is used to modify the table structure.

The following describes in detail what happens when you use the DELAYED option to INSERT or REPLACE. In this description, the “thread” is the thread that received an INSERT DELAYED statement and “handler” is the thread that handles all INSERT DELAYED statements for a particular table.

- When a thread executes a DELAYED statement for a table, a handler thread is created to process all DELAYED statements for the table, if no such handler already exists.
- The thread checks whether the handler has previously acquired a DELAYED lock; if not, it tells the handler thread to do so. The DELAYED lock can be obtained even if other threads have a READ or WRITE lock on the table. However, the handler waits for all ALTER TABLE locks or FLUSH TABLES statements to finish, to ensure that the table structure is up to date.
- The thread executes the INSERT statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.
- The client cannot obtain from the server the number of duplicate rows or the AUTO_INCREMENT value for the resulting row, because the INSERT returns before the insert operation has been completed. (If you use the C API, the mysql_info() function does not return anything meaningful, for the same reason.)
- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- Each time that delayed_insert_limit [410] rows are written, the handler checks whether any SELECT statements are still pending. If so, it permits these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new INSERT DELAYED statements are received within delayed_insert_timeout [410] seconds, the handler terminates.
- If more than delayed_queue_size [410] rows are pending in a specific handler queue, the thread requesting INSERT DELAYED waits until there is room in the queue. This is done to ensure that mysqld does not use all memory for the delayed memory queue.
- The handler thread shows up in the MySQL process list with delayed_insert in the Command column. It is killed if you execute a FLUSH TABLES statement or kill it with KILL *thread_id*. However, before exiting, it first stores all queued rows into the table. During this time it does not accept any new INSERT statements from other threads. If you execute an INSERT DELAYED statement after this, a new handler thread is created.

  Note that this means that INSERT DELAYED statements have higher priority than normal INSERT statements if there is an INSERT DELAYED handler running. Other update statements have to wait until the INSERT DELAYED queue is empty, someone terminates the handler thread (with KILL *thread_id*), or someone executes a FLUSH TABLES.
- The following status variables provide information about INSERT DELAYED statements.

Status Variable	Meaning
Delayed_insert_threads [451]	Number of handler threads
Delayed_writes [451]	Number of rows written with INSERT DELAYED
Not_flushed_delayed_rows	Number of rows waiting to be written

  You can view these variables by issuing a SHOW STATUS statement or by executing a mysqladmin extended-status command.

12.2.4.3 INSERT ... ON DUPLICATE KEY UPDATE Syntax

If you specify ON DUPLICATE KEY UPDATE (added in MySQL 4.1.0), and a row is inserted that would cause a duplicate value in a UNIQUE index or PRIMARY KEY, an UPDATE of the old row is performed. For example, if column a is declared as UNIQUE and contains the value 1, the following two statements have identical effect:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=c+1;

UPDATE table SET c=c+1 WHERE a=1;
```

With ON DUPLICATE KEY UPDATE, the affected-rows value per row is 1 if the row is inserted as a new row and 2 if an existing row is updated.

If column b is also unique, the INSERT is equivalent to this UPDATE statement instead:

```
UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If a=1 OR b=2 matches several rows, only *one* row is updated. In general, you should try to avoid using an ON DUPLICATE KEY UPDATE clause on tables with multiple unique indexes.

The ON DUPLICATE KEY UPDATE clause can contain multiple column assignments, separated by commas.

As of MySQL 4.1.1, you can use the VALUES(*col_name*) [880] function in the UPDATE clause to refer to column values from the INSERT portion of the INSERT ... ON DUPLICATE KEY UPDATE statement. In other words, VALUES(*col_name*) [880] in the ON DUPLICATE KEY UPDATE clause refers to the value of *col_name* that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The VALUES() [880] function is meaningful only in INSERT ... UPDATE statements and returns NULL otherwise. Example:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
  ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=3;
INSERT INTO table (a,b,c) VALUES (4,5,6)
  ON DUPLICATE KEY UPDATE c=9;
```

If a table contains an AUTO_INCREMENT column and INSERT ... ON DUPLICATE KEY UPDATE inserts a row, the LAST_INSERT_ID() [874] function returns the AUTO_INCREMENT value. If the statement updates a row instead, LAST_INSERT_ID() [874] is not meaningful. However, you can work around this by using LAST_INSERT_ID(*expr*) [874]. Suppose that id is the AUTO_INCREMENT column. To make LAST_INSERT_ID() [874] meaningful for updates, insert rows as follows:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE id=LAST_INSERT_ID(id), c=3;
```

The DELAYED option is ignored when you use ON DUPLICATE KEY UPDATE.

12.2.5 LOAD DATA INFILE Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
```

```
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number LINES]
    [(col_name,...)]
```

The LOAD DATA INFILE statement reads rows from a text file into a table at a very high speed. The file name must be given as a literal string.

LOAD DATA INFILE is the complement of SELECT ... INTO OUTFILE. (See Section 12.2.7, "SELECT Syntax".) To write data from a table to a file, use SELECT ... INTO OUTFILE. To read the file back into a table, use LOAD DATA INFILE. The syntax of the FIELDS and LINES clauses is the same for both statements. Both clauses are optional, but FIELDS must precede LINES if both are specified.

For more information about the efficiency of INSERT versus LOAD DATA INFILE and speeding up LOAD DATA INFILE, see Section 7.3.2.1, "Speed of INSERT Statements".

As of MySQL 4.1, the character set indicated by the character_set_database [408] system variable is used to interpret the information in the file. SET NAMES and the setting of the character_set_client [408] system variable do not affect interpretation of input.

LOAD DATA INFILE interprets all fields in the file as having the same character set, regardless of the data types of the columns into which field values are loaded. For proper interpretation of file contents, you must ensure that it was written with the correct character set. For example, if you write a data file with mysqldump -T or by issuing a SELECT ... INTO OUTFILE statement in mysql, be sure to use a --default-character-set option with mysqldump or mysql so that output is written in the character set to be used when the file is loaded with LOAD DATA INFILE.

Note that it is currently not possible to load data files that use the ucs2 character set.

You can also load data files by using the mysqlimport utility; it operates by sending a LOAD DATA INFILE statement to the server. The --local [304] option causes mysqlimport to read data files from the client host. You can specify the --compress [303] option to get better performance over slow networks if the client and server support the compressed protocol. See Section 4.5.5, "mysqlimport — A Data Import Program".

If you use LOW_PRIORITY, execution of the LOAD DATA statement is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

If you specify CONCURRENT with a MyISAM table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while LOAD DATA is executing. Using this option affects the performance of LOAD DATA a bit, even if no other thread is using the table at the same time.

CONCURRENT is not replicated. See Section 14.7, "Replication Features and Issues", for more information.

The LOCAL keyword, if specified, is interpreted with respect to the client end of the connection:

- If LOCAL is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full path name to specify its exact location. If given as a relative path name, the name is interpreted relative to the directory in which the client program was started.

  LOCAL is available in MySQL 3.22.6 or later.

- If LOCAL is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:
  - If the file name is an absolute path name, the server uses it as given.
  - If the file name is a relative path name with one or more leading components, the server searches for the file relative to the server's data directory.
  - If a file name with no leading components is given, the server looks for the file in the database directory of the default database.

Note that, in the non-LOCAL case, these rules mean that a file named as ./myfile.txt is read from the server's data directory, whereas the file named as myfile.txt is read from the database directory of the default database. For example, if db1 is the default database, the following LOAD DATA statement reads the file data.txt from the database directory for db1, even though the statement explicitly loads the file into a table in the db2 database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Windows path names are specified using forward slashes rather than backslashes. If you do use backslashes, you must double them.

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use LOAD DATA INFILE on server files, you must have the FILE [491] privilege. See Section 5.5.1, “Privileges Provided by MySQL”.

Using LOCAL is a bit slower than letting the server access the files directly, because the contents of the file must be sent over the connection by the client to the server. On the other hand, you do not need the FILE [491] privilege to load local files.

With LOCAL, the default duplicate-key handling behavior is the same as if IGNORE is specified; this is because the server has no way to stop transmission of the file in the middle of the operation. IGNORE is explained further later in this section.

As of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows), LOCAL works only if your server and your client both have been configured to permit it. For example, if mysqld was started with --local-infile=0 [486], LOCAL does not work. See Section 5.4.5, “Security Issues with LOAD DATA LOCAL”.

On Unix, if you need LOAD DATA to read from a pipe, you can use the following technique (the example loads a listing of the / directory into the table db1.t1):

```
mkfifo /mysql/data/db1/ls.dat
chmod 666 /mysql/data/db1/ls.dat
find / -ls > /mysql/data/db1/ls.dat &
mysql -e "LOAD DATA INFILE 'ls.dat' INTO TABLE t1" db1
```

Note that you must run the command that generates the data to be loaded and the mysql commands either on separate terminals, or run the data generation process in the background (as shown in the preceding example). If you do not do this, the pipe will block until data is read by the mysql process.

If you are using a version of MySQL older than 3.23.25, you can use this technique only with LOAD DATA LOCAL INFILE.

If you are using MySQL before version 3.23.24, you cannot read from a FIFO with LOAD DATA INFILE. If you need to read from a FIFO (for example, the output from gunzip), use LOAD DATA LOCAL INFILE instead.

The REPLACE and IGNORE keywords control handling of input rows that duplicate existing rows on unique key values:

- If you specify REPLACE, input rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row. See Section 12.2.6, “REPLACE Syntax”.
- If you specify IGNORE, input rows that duplicate an existing row on a unique key value are skipped.
- If you do not specify either option, the behavior depends on whether the LOCAL keyword is specified. Without LOCAL, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With LOCAL, the default behavior is the same as if IGNORE is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

To ignore foreign key constraints during the load operation, issue a SET foreign_key_checks = 0 statement before executing LOAD DATA.

If you use LOAD DATA INFILE on an empty MyISAM table, all nonunique indexes are created in a separate batch (as for REPAIR TABLE). Normally, this makes LOAD DATA INFILE much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with ALTER TABLE ... DISABLE KEYS before loading the file into the table and using ALTER TABLE ... ENABLE KEYS to re-create the indexes after loading the file. See Section 7.3.2.1, “Speed of INSERT Statements”.

For both the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements, the syntax of the FIELDS and LINES clauses is the same. Both clauses are optional, but FIELDS must precede LINES if both are specified.

If you specify a FIELDS clause, each of its subclauses (TERMINATED BY, [OPTIONALLY] ENCLOSED BY, and ESCAPED BY) is also optional, except that you must specify at least one of them.

If you specify no FIELDS or LINES clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

(Backslash is the MySQL escape character within strings in SQL statements, so to specify a literal backslash, you must specify two backslashes for the value to be interpreted as a single backslash. The escape sequences '\t' and '\n' specify tab and newline characters, respectively.)

In other words, the defaults cause LOAD DATA INFILE to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip over any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret characters preceded by the escape character “\” as escape sequences. For example, “\t”, “\n”, and “\\” signify tab, newline, and backslash, respectively. See the discussion of FIELDS ESCAPED BY later for the full list of escape sequences.

Conversely, the defaults cause SELECT ... INTO OUTFILE to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.

- Use "\" to escape instances of tab, newline, or "\" that occur within field values.
- Write newlines at the ends of lines.

Note

If you have generated the text file on a Windows system, you might have to use `LINES TERMINATED BY '\r\n'` to read the file properly, because Windows programs typically use two characters as a line terminator. Some programs, such as `WordPad`, might use `\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\r'`.

If all the lines you want to read in have a common prefix that you want to ignore, you can use `LINES STARTING BY 'prefix_string'` to skip over the prefix, *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
  FIELDS TERMINATED BY ','  LINES STARTING BY 'xxx';
```

If the data file looks like this:

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows will be `("abc",1)` and `("def",2)`. The third row in the file is skipped because it does not contain the prefix.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip over an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA INFILE` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA INFILE` will not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
  FROM table2;
```

To read the comma-delimited file back in, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown following, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

LOAD DATA INFILE can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated by commas and enclosed within double quotation marks, with an initial line of column names. If the lines in such a file are terminated by carriage return/newline pairs, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

If the input values are not necessarily enclosed within quotation marks, use OPTIONALLY before the ENCLOSED BY keywords.

Any of the field- or line-handling options can specify an empty string (''). If not empty, the FIELDS [OPTIONALLY] ENCLOSED BY and FIELDS ESCAPED BY values must be a single character. The FIELDS TERMINATED BY, LINES STARTING BY, and LINES TERMINATED BY values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a LINES TERMINATED BY '\r\n' clause.

To read a file containing jokes that are separated by lines consisting of %%, you can do this

```
CREATE TABLE jokes
  (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ''
  LINES TERMINATED BY '\n%%\n' (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY controls quoting of fields. For output (SELECT ... INTO OUTFILE), if you omit the word OPTIONALLY, all fields are enclosed by the ENCLOSED BY character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify OPTIONALLY, the ENCLOSED BY character is used only to enclose values from columns that have a string data type (such as CHAR, BINARY, TEXT, or ENUM):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the ENCLOSED BY character within a field value are escaped by prefixing them with the ESCAPED BY character. Also note that if you specify an empty ESCAPED BY value, it is possible to inadvertently generate output that cannot be read properly by LOAD DATA INFILE. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
```

```
4,"a string containing a ", quote and comma",102.20
```

For input, the ENCLOSED BY character, if present, is stripped from the ends of field values. (This is true regardless of whether OPTIONALLY is specified; OPTIONALLY has no effect on input interpretation.) Occurrences of the ENCLOSED BY character preceded by the ESCAPED BY character are interpreted as part of the current field value.

If the field begins with the ENCLOSED BY character, instances of that character are recognized as terminating a field value only if followed by the field or line TERMINATED BY sequence. To avoid ambiguity, occurrences of the ENCLOSED BY character within a field value can be doubled and are interpreted as a single instance of the character. For example, if ENCLOSED BY '"' is specified, quotation marks are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY controls how to read or write special characters:

- For input, if the FIELDS ESCAPED BY character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. Some two-character sequences that are exceptions, where the first character is the escape character. These sequences are shown in the following table (using "\" for the escape character). The rules for NULL handling are described later in this section.

Character	Escape Sequence
\0	An ASCII NUL (0x00) character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character.
\Z	ASCII 26 (Control-Z)
\N	NULL

  For more information about "\"-escape syntax, see Section 8.1.1, "String Literals".

  If the FIELDS ESCAPED BY character is empty, escape-sequence interpretation does not occur.

- For output, if the FIELDS ESCAPED BY character is not empty, it is used to prefix the following characters on output:

  - The FIELDS ESCAPED BY character
  - The FIELDS [OPTIONALLY] ENCLOSED BY character
  - The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values
  - ASCII 0 (what is actually written following the escape character is ASCII "0", not a zero-valued byte)

  If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

In certain cases, field- and line-handling options interact:

- If LINES TERMINATED BY is an empty string and FIELDS TERMINATED BY is nonempty, lines are also terminated with FIELDS TERMINATED BY.
- If the FIELDS TERMINATED BY and FIELDS ENCLOSED BY values are both empty (''), a fixed-row (nondelimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are read and written using a field width wide enough to hold all values in the field. For TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

  LINES TERMINATED BY is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to ''. In this case, the text file must contain all fields for each row.

  Fixed-row format also affects handling of NULL values, as described later. Note that fixed-size format does not work if you are using a multi-byte character set.

Note

Before MySQL 4.1.12, fixed-row format used the display width of the column. For example, INT(4) was read or written using a field with a width of 4. However, if the column contained wider values, they were dumped to their full width, leading to the possibility of a “ragged” field holding values of different widths. Using a field wide enough to hold all values in the field prevents this problem. However, data files written before this change was made might not be reloaded correctly with LOAD DATA INFILE for MySQL 4.1.12 and up. This change also affects data files read by mysqlimport and written by mysqldump --tab, which use LOAD DATA INFILE and SELECT ... INTO OUTFILE.

Handling of NULL values varies according to the FIELDS and LINES options in use:

- For the default FIELDS and LINES values, NULL is written as a field value of \N for output, and a field value of \N is read as NULL for input (assuming that the ESCAPED BY character is “\”).
- If FIELDS ENCLOSED BY is not empty, a field containing the literal word NULL as its value is read as a NULL value. This differs from the word NULL enclosed within FIELDS ENCLOSED BY characters, which is read as the string 'NULL'.
- If FIELDS ESCAPED BY is empty, NULL is written as the word NULL.
- With fixed-row format (which is used when FIELDS TERMINATED BY and FIELDS ENCLOSED BY are both empty), NULL is written as an empty string. Note that this causes both NULL values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load NULL into a NOT NULL column causes assignment of the implicit default value for the column's data type and a warning. Implicit default values are discussed in Section 10.1.4, “Data Type Default Values”.

Some cases are not supported by LOAD DATA INFILE:

- Fixed-size rows (FIELDS TERMINATED BY and FIELDS ENCLOSED BY both empty) and BLOB or TEXT columns.
- If you specify one separator that is the same as or a prefix of another, LOAD DATA INFILE cannot interpret the input properly. For example, the following FIELDS clause would cause problems:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- If FIELDS ESCAPED BY is empty, a field value that contains an occurrence of FIELDS ENCLOSED BY or LINES TERMINATED BY followed by the FIELDS TERMINATED BY value causes LOAD DATA INFILE to stop reading a field or line too early. This happens because LOAD DATA INFILE cannot properly determine where the field or line value ends.

The following example loads all columns of the persondata table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the LOAD DATA INFILE statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2,...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in Section 10.1.4, “Data Type Default Values”.

An empty field value is interpreted different from a missing field:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to 0.
- For date and time types, the column is set to the appropriate “zero” value for the type. See Section 10.3, “Date and Time Types”.

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an INSERT or UPDATE statement.

TIMESTAMP columns are set to the current date and time only if there is a NULL value for the column (that is, \N) and the column is not declared to permit NULL values, or if the TIMESTAMP column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

LOAD DATA INFILE regards all input as strings, so you cannot use numeric values for ENUM or SET columns the way you can with INSERT statements. All ENUM and SET values must be specified as strings.

When the LOAD DATA INFILE statement finishes, it returns an information string in the following format:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Warnings occur under the same circumstances as when values are inserted using the INSERT statement (see Section 12.2.4, “INSERT Syntax”), except that LOAD DATA INFILE also generates warnings when there are too few or too many fields in the input row. The warnings are not stored anywhere; the number of warnings can be used only as an indication of whether everything went well.

From MySQL 4.1.1 on, you can use SHOW WARNINGS to get a list of the first max_error_count [419] warnings as information about what went wrong. See Section 12.4.5.26, “SHOW WARNINGS Syntax”.

If you are using the C API, you can get information about the statement by calling the mysql_info() function. See Section 17.6.6.33, “mysql_info()”.

Before MySQL 4.1.1, only a warning count is available to indicate that something went wrong. If you get warnings and want to know exactly why you got them, one way to do this is to dump the table into another file using `SELECT ... INTO OUTFILE` and compare the file to your original input file.

12.2.6 REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

`REPLACE` works exactly like `INSERT`, except that if an old row in the table has the same value as a new row for a `PRIMARY KEY` or a `UNIQUE` index, the old row is deleted before the new row is inserted. See Section 12.2.4, “`INSERT` Syntax”.

`REPLACE` is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL—that either inserts or *updates*—see Section 12.2.4.3, “`INSERT ... ON DUPLICATE KEY UPDATE` Syntax”. `INSERT ... ON DUPLICATE KEY UPDATE` is available as of MySQL 4.1.0.

Note that unless the table has a `PRIMARY KEY` or `UNIQUE` index, using a `REPLACE` statement makes no sense. It becomes equivalent to `INSERT`, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the `REPLACE` statement. Any missing columns are set to their default values, just as happens for `INSERT`. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as `SET col_name = col_name + 1`, the reference to the column name on the right hand side is treated as `DEFAULT(col_name)` [877], so the assignment is equivalent to `SET col_name = DEFAULT(col_name) + 1`.

To use `REPLACE`, you must have both the `INSERT` [492] and `DELETE` [491] privileges for the table.

The `REPLACE` statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row `REPLACE`, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether `REPLACE` only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the `mysql_affected_rows()` function.

Currently, you cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for REPLACE (and LOAD DATA ... REPLACE):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
   a. Delete from the table the conflicting row that has the duplicate key value
   b. Try again to insert the new row into the table

It is possible that in the case of a duplicate-key error, a storage engine may perform the REPLACE as an update rather than a delete plus insert, but the semantics are the same. There are no user-visible effects other than a possible difference in how the storage engine increments Handler_*xxx* status variables.

12.2.7 SELECT Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name'
      | INTO @var_name [, @var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT is used to retrieve rows selected from one or more tables. Support for UNION statements and subqueries is available as of MySQL 4.0 and 4.1, respectively. See Section 12.2.7.3, “UNION Syntax”, and Section 12.2.8, “Subquery Syntax”.

The most commonly used clauses of SELECT statements are these:

- Each *select_expr* indicates a column that you want to retrieve. There must be at least one *select_expr*.
- *table_references* indicates the table or tables from which to retrieve rows. Its syntax is described in Section 12.2.7.1, “JOIN Syntax”.
- The WHERE clause, if given, indicates the condition or conditions that rows must satisfy to be selected. *where_condition* is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no WHERE clause.

  In the WHERE expression, you can use any of the functions and operators that MySQL supports, except for aggregate (summary) functions. See Section 8.5, “Expression Syntax”, and Chapter 11, *Functions and Operators*.

SELECT can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
       -> 2
```

From MySQL 4.1.0 on, you are permitted to specify DUAL as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;
       -> 2
```

DUAL is purely for the convenience of people who require that all SELECT statements should have FROM and possibly other clauses. MySQL may ignore the clauses. MySQL does not require FROM DUAL if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a HAVING clause must come after any GROUP BY clause and before any ORDER BY clause. The exception is that the INTO clause can appear either as shown in the syntax description or immediately following the *select_expr* list.

The list of *select_expr* terms comprises the select list that indicates which columns to retrieve. Terms specify a column or expression or can use *-shorthand:

- A select list consisting only of a single unqualified * can be used as shorthand to select all columns from all tables:

  ```
  SELECT * FROM t1 INNER JOIN t2 ...
  ```

- *tbl_name*.* can be used as a qualified shorthand to select all columns from the named table:

  ```
  SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
  ```

- Use of an unqualified * with other items in the select list may produce a parse error. To avoid this problem, use a qualified *tbl_name*.* reference

  ```
  SELECT AVG(score), t1.* FROM t1 ...
  ```

The following list provides additional information about other SELECT clauses:

- A *select_expr* can be given an alias using AS *alias_name*. The alias is used as the expression's column name and can be used in GROUP BY, ORDER BY, or HAVING clauses. For example:

  ```
  SELECT CONCAT(last_name,', ',first_name) AS full_name
    FROM mytable ORDER BY full_name;
  ```

  The AS keyword is optional when aliasing a *select_expr* with an identifier. The preceding example could have been written like this:

  ```
  SELECT CONCAT(last_name,', ',first_name) full_name
    FROM mytable ORDER BY full_name;
  ```

  However, because the AS is optional, a subtle problem can occur if you forget the comma between two *select_expr* expressions: MySQL interprets the second as an alias name. For example, in the following statement, columnb is treated as an alias name:

  ```
  ```

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using AS explicitly when specifying column aliases.

It is not permissible to refer to a column alias in a WHERE clause, because the column value might not yet be determined when the WHERE clause is executed. See Section B.5.5.4, “Problems with Column Aliases”.

- The FROM *table_references* clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see Section 12.2.7.1, “JOIN Syntax”. For each table specified, you can optionally specify an alias.

```
tbl_name [[AS] alias] [index_hint)]
```

The use of index hints provides the optimizer with information about how to choose indexes during query processing. For a description of the syntax for specifying these hints, see Section 12.2.7.2, “Index Hint Syntax”.

In MySQL 4.0.14, you can use SET max_seeks_for_key=*value* as an alternative way to force MySQL to prefer key scans instead of table scans. See Section 5.1.3, “Server System Variables”.

- You can refer to a table within the default database as *tbl_name*, or as *db_name*.*tbl_name* to specify a database explicitly. You can refer to a column as *col_name*, *tbl_name*.*col_name*, or *db_name*.*tbl_name*.*col_name*. You need not specify a *tbl_name* or *db_name*.*tbl_name* prefix for a column reference unless the reference would be ambiguous. See Section 8.2.1, “Identifier Qualifiers”, for examples of ambiguity that require the more explicit column reference forms.
- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name* *alias_name*:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  WHERE t1.name = t2.name;

SELECT t1.name, t2.salary FROM employee t1, info t2
  WHERE t1.name = t2.name;
```

- Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

```
SELECT college, region, seed FROM tournament
  ORDER BY region, seed;

SELECT college, region AS r, seed AS s FROM tournament
  ORDER BY r, s;

SELECT college, region, seed FROM tournament
  ORDER BY 2, 3;
```

To sort in reverse order, add the DESC (descending) keyword to the name of the column in the ORDER BY clause that you are sorting by. The default is ascending order; this can be specified explicitly using the ASC keyword.

If ORDER BY occurs within a subquery and also is applied in the outer query, the outermost ORDER BY takes precedence. For example, results for the following statement are sorted in descending order, not ascending order:

```
(SELECT ... ORDER BY a) ORDER BY a DESC;
```

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- If you use GROUP BY, output rows are sorted according to the GROUP BY columns as if you had an ORDER BY for the same columns. To avoid the overhead of sorting that GROUP BY produces, add ORDER BY NULL:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- MySQL extends the GROUP BY clause as of version 3.23.34 so that you can also specify ASC and DESC after columns named in the clause:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
```

- MySQL extends the use of GROUP BY to permit selecting fields that are not mentioned in the GROUP BY clause. If you are not getting the results that you expect from your query, please read the description of GROUP BY found in Section 11.15, “Functions and Modifiers for Use with GROUP BY Clauses”.
- As of MySQL 4.1.1, GROUP BY permits a WITH ROLLUP modifier. See Section 11.15.2, “GROUP BY Modifiers”.
- The HAVING clause is applied nearly last, just before items are sent to the client, with no optimization. (LIMIT is applied after HAVING.)

  A HAVING clause can refer to any column or alias named in a *select_expr* in the SELECT list or in outer subqueries, and to aggregate functions. (Standard SQL requires that HAVING must reference only columns in the GROUP BY clause or columns used in aggregate functions.)
- Do not use HAVING for items that should be in the WHERE clause. For example, do not write the following:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

  Write this instead:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- The HAVING clause can refer to aggregate functions, which the WHERE clause cannot:

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

  However, that does not work in older MySQL servers (before version 3.22.5). In those versions, you can use a column alias in the select list and refer to the alias in the HAVING clause:

```
SELECT user, MAX(salary) AS max_salary FROM users
  GROUP BY user HAVING max_salary>10;
```

- MySQL permits duplicate column names. That is, there can be more than one *select_expr* with the same name. This is an extension to standard SQL. Because MySQL also permits GROUP BY and HAVING to refer to *select_expr* values, this can result in an ambiguity:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

In that statement, both columns have the name a. To ensure that the correct column is used for grouping, use different names for each *select_expr*.

- When MySQL resolves an unqualified column or alias reference in an ORDER BY, GROUP BY, or HAVING clause, it first searches for the name in the *select_expr* values. If the name is not found, it looks in the columns of the tables named in the FROM clause.

- The LIMIT clause can be used to constrain the number of rows returned by the SELECT statement. LIMIT takes one or two numeric arguments, which must both be nonnegative integer constants (except when using prepared statements).

  With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

  ```
  SELECT * FROM tbl LIMIT 5,10;  # Retrieve rows 6-15
  ```

  To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

  ```
  SELECT * FROM tbl LIMIT 95,18446744073709551615;
  ```

  With one argument, the value specifies the number of rows to return from the beginning of the result set:

  ```
  SELECT * FROM tbl LIMIT 5;     # Retrieve first 5 rows
  ```

  In other words, LIMIT *row_count* is equivalent to LIMIT 0, *row_count*.

  For prepared statements, you can use placeholders (supported as of MySQL version 5.0.7). The following statements will return one row from the tbl table:

  ```
  SET @a=1;
  PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
  EXECUTE STMT USING @a;
  ```

  The following statements will return the second to sixth row from the tbl table:

  ```
  SET @skip=1; SET @numrows=5;
  PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
  EXECUTE STMT USING @skip, @numrows;
  ```

  For compatibility with PostgreSQL, MySQL also supports the LIMIT *row_count* OFFSET *offset* syntax.

  If LIMIT occurs within a subquery and also is applied in the outer query, the outermost LIMIT takes precedence. For example, the following statement produces two rows, not one:

  ```
  (SELECT ... LIMIT 1) LIMIT 2;
  ```

- A PROCEDURE clause names a procedure that should process the data in the result set. For an example, see Section 18.3.1, “PROCEDURE ANALYSE”, which describes ANALYSE, a procedure that can be used to obtain suggestions for optimal column data types that may help reduce table sizes.

- The SELECT ... INTO OUTFILE '*file_name*' form of SELECT writes the selected rows to a file. The file is created on the server host, so you must have the FILE [491] privilege to use this syntax.

file_name cannot be an existing file, which among other things prevents files such as /etc/passwd and database tables from being destroyed.

The SELECT ... INTO OUTFILE statement is intended primarily to let you very quickly dump a table to a text file on the server machine. If you want to create the resulting file on some other host than the server host, you normally cannot use SELECT ... INTO OUTFILE since there is no way to write a path to the file relative to the server host's file system.

However, if the MySQL client software is installed on the remote machine, you can instead use a client command such as mysql -e "SELECT ..." > *file_name* to generate the file on the client host.

It is also possible to create the resulting file on a different host other than the server host, if the location of the file on the remote host can be accessed using a network-mapped path on the server's file system. In this case, the presence of mysql (or some other MySQL client program) is not required on the target host.

SELECT ... INTO OUTFILE is the complement of LOAD DATA INFILE. Column values are dumped using the binary character set. In effect, there is no character set conversion. If a table contains columns in several character sets, the output data file will as well and you may not be able to reload the file correctly.

The syntax for the *export_options* part of the statement consists of the same FIELDS and LINES clauses that are used with the LOAD DATA INFILE statement. See Section 12.2.5, “LOAD DATA INFILE Syntax”, for information about the FIELDS and LINES clauses, including their default values and permissible values.

FIELDS ESCAPED BY controls how to write special characters. If the FIELDS ESCAPED BY character is not empty, it is used as a prefix that precedes following characters on output:

- The FIELDS ESCAPED BY character
- The FIELDS [OPTIONALLY] ENCLOSED BY character
- The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values
- ASCII NUL (the zero-valued byte; what is actually written following the escape character is ASCII “0”, not a zero-valued byte)

The FIELDS TERMINATED BY, ENCLOSED BY, ESCAPED BY, or LINES TERMINATED BY characters *must* be escaped so that you can read the file back in reliably. ASCII NUL is escaped to make it easier to view with some pagers.

The resulting file does not have to conform to SQL syntax, so nothing else need be escaped.

If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
  FROM test_table;
```

- If you use INTO DUMPFILE instead of INTO OUTFILE, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a BLOB value in a file.

- 

  Note

  Any file created by INTO OUTFILE or INTO DUMPFILE is writable by all users on the server host. The reason for this is that the MySQL server cannot create a file that is owned by anyone other than the user under whose account it is running. (You should *never* run mysqld as root for this and other reasons.) The file thus must be world-writable so that you can manipulate its contents.

- As of MySQL 4.1, the INTO clause can name a list of one or more user-defined variables. The selected values are assigned to the variables. The number of variables must match the number of columns. The query should return a single row. If the query returns no rows, error 1065 occurs (Query was empty). If the query returns multiple rows, error 1172 occurs (Result consisted of more than one row).

- The SELECT syntax description at the beginning this section shows the INTO clause near the end of the statement. It is also possible to use INTO immediately following the *select_expr* list.

- An INTO clause should not be used in a nested SELECT because such a SELECT must return its result to the outer context.

- If you use FOR UPDATE with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using LOCK IN SHARE MODE sets a shared lock that permits other transactions to read the examined rows but not to update or delete them. See Section 13.2.9.3, "SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads".

Following the SELECT keyword, you can use a number of options that affect the operation of the statement. HIGH_PRIORITY, STRAIGHT_JOIN, and options beginning with SQL_ are MySQL extensions to standard SQL.

- The ALL and DISTINCT options specify whether duplicate rows should be returned. ALL (the default) specifies that all matching rows should be returned, including duplicates. DISTINCT specifies removal of duplicate rows from the result set. As of MySQL 4.1, it is an error to specify both options. DISTINCTROW is a synonym for DISTINCT.

- HIGH_PRIORITY gives the SELECT higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A SELECT HIGH_PRIORITY query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

  HIGH_PRIORITY cannot be used with SELECT statements that are part of a UNION.

- STRAIGHT_JOIN forces the optimizer to join the tables in the order in which they are listed in the FROM clause. You can use this to speed up a query if the optimizer joins the tables in nonoptimal order. STRAIGHT_JOIN also can be used in the *table_references* list. See Section 12.2.7.1, "JOIN Syntax".

  STRAIGHT_JOIN does not apply to any table that the optimizer treats as a const [567] or system [566] table. Such a table produces a single row, is read during the optimization phase of query execution, and references to its columns are replaced with the appropriate column values before query execution proceeds. These tables will appear first in the query plan displayed by EXPLAIN. See Section 7.2.1, "Optimizing Queries with EXPLAIN". This exception may not apply to const [567] or

system [566] tables that are used on the NULL-complemented side of an outer join (that is, the right-side table of a LEFT JOIN or the left-side table of a RIGHT JOIN.

- SQL_BIG_RESULT or SQL_SMALL_RESULT can be used with GROUP BY or DISTINCT to tell the optimizer that the result set has many rows or is small, respectively. For SQL_BIG_RESULT, MySQL directly uses disk-based temporary tables if needed, and prefers sorting to using a temporary table with a key on the GROUP BY elements. For SQL_SMALL_RESULT, MySQL uses fast temporary tables to store the resulting table instead of using sorting. This should not normally be needed.

- SQL_BUFFER_RESULT forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client. This option can be used only for top-level SELECT statements, not for subqueries or following UNION.

- SQL_CALC_FOUND_ROWS (available in MySQL 4.0.0 and up) tells MySQL to calculate how many rows there would be in the result set, disregarding any LIMIT clause. The number of rows can then be retrieved with SELECT FOUND_ROWS(). See Section 11.13, "Information Functions".

  Before MySQL 4.1.0, this option does not work with LIMIT 0, which is optimized to return instantly (resulting in a row count of 0). See Section 7.3.1.10, "LIMIT Optimization".

- The SQL_CACHE and SQL_NO_CACHE options affect caching of query results in the query cache (see Section 7.5.3, "The MySQL Query Cache"). SQL_CACHE tells MySQL to store the result in the query cache if it is cacheable and the value of the query_cache_type [424] system variable is 2 or DEMAND. SQL_NO_CACHE tells MySQL not to store the result in the query cache. For a query that uses UNION or subqueries, the following rules apply:

  - SQL_NO_CACHE applies if it appears in any SELECT in the query.

  - For a cacheable query, SQL_CACHE applies if it appears in the first SELECT of the query.

12.2.7.1 JOIN Syntax

MySQL supports the following JOIN syntaxes for the *table_references* part of SELECT statements and multiple-table DELETE and UPDATE statements:

```
table_references:
    table_reference, table_reference
  | table_reference [INNER | CROSS] JOIN table_reference [join_condition]
  | table_reference STRAIGHT_JOIN table_reference
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_reference
  | { OJ table_reference LEFT OUTER JOIN table_reference
        ON conditional_expr }

table_reference:
    tbl_name [[AS] alias] [index_hint)]
  | table_subquery [AS] alias

join_condition:
    ON conditional_expr
  | USING (column_list)

index_hint:
    USE {INDEX|KEY} (index_list)]
  | IGNORE {INDEX|KEY} (index_list)]
  | FORCE {INDEX|KEY} (index_list)]

index_list:
    index_name [, index_name] ...
```

Index hints can be specified to affect how the MySQL optimizer makes use of indexes. For more information, see Section 12.2.7.2, “Index Hint Syntax”.

Note that several changes in join processing were made in MySQL 5.0.12 to make MySQL more compliant with standard SQL. These changes include the ability to handle nested joins (including outer joins) according to the standard. If a nested join returns results that are not what you expect, please consider upgrading to MySQL 5.0. Further details about the changes in join processing can be found at JOIN Syntax.

You should generally not have any conditions in the ON part that are used to restrict which rows you want in the result set, but rather specify these conditions in the WHERE clause. There are exceptions to this rule.

Note that INNER JOIN syntax permits a join_condition only from MySQL 3.23.17 on. The same is true for JOIN and CROSS JOIN only as of MySQL 4.0.11.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name alias_name*:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  WHERE t1.name = t2.name;

SELECT t1.name, t2.salary FROM employee t1, info t2
  WHERE t1.name = t2.name;
```

- A *table_subquery* is also known as a subquery in the FROM clause. Such subqueries *must* include an alias to give the subquery result a table name. A trivial example follows; see also Section 12.2.8.8, “Subqueries in the FROM Clause”.

```
SELECT * FROM (SELECT 1, 2, 3) AS t1;
```

- The *conditional_expr* used with ON is any conditional expression of the form that can be used in a WHERE clause.

- If there is no matching row for the right table in the ON or USING part in a LEFT JOIN, a row with all columns set to NULL is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

```
SELECT left_tbl.*
  FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
  WHERE right_tbl.id IS NULL;
```

  This example finds all rows in left_tbl with an id value that is not present in right_tbl (that is, all rows in left_tbl with no corresponding row in right_tbl). This assumes that right_tbl.id is declared NOT NULL. See Section 7.3.1.5, “LEFT JOIN and RIGHT JOIN Optimization”.

- The USING(*column_list*) clause names a list of columns that must exist in both tables. The following two clauses are semantically identical:

```
a LEFT JOIN b USING (c1,c2,c3)
a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
```

- The NATURAL [LEFT] JOIN of two tables is defined to be semantically equivalent to an INNER JOIN or a LEFT JOIN with a USING clause that names all columns that exist in both tables.

- INNER JOIN and , (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

- RIGHT JOIN works analogously to LEFT JOIN. To keep code portable across databases, it is recommended that you use LEFT JOIN instead of RIGHT JOIN.
-
- STRAIGHT_JOIN is similar to JOIN, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

Some join examples:

```
SELECT * FROM table1,table2 WHERE table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 USING (id);

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  LEFT JOIN table3 ON table2.id=table3.id;
```

12.2.7.2 Index Hint Syntax

As of MySQL 3.23.12, you can provide hints to give the optimizer information about how to choose indexes during query processing. Section 12.2.7.1, "JOIN Syntax", describes the general syntax for specifying tables in a SELECT statement. The syntax for an individual table, including that for index hints, looks like this:

```
tbl_name [[AS] alias] [index_hint)]

index_hint:
    USE {INDEX|KEY} (index_list)]
  | IGNORE {INDEX|KEY} (index_list)]
  | FORCE {INDEX|KEY} (index_list)]

index_list:
    index_name [, index_name] ...
```

By specifying USE INDEX (*index_list*), you can tell MySQL to use only one of the named indexes to find rows in the table. The alternative syntax IGNORE INDEX (*index_list*) can be used to tell MySQL to not use some particular index or indexes. These hints are useful if EXPLAIN shows that MySQL is using the wrong index from the list of possible indexes.

From MySQL 4.0.9 on, you can also use FORCE INDEX, which acts like USE INDEX (*index_list*) but with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is used only if there is no way to use one of the given indexes to find rows in the table.

Each hint requires the names of *indexes*, not the names of columns. The name of a PRIMARY KEY is PRIMARY. To see the index names for a table, use SHOW INDEX.

An *index_name* value need not be a full index name. It can be an unambiguous prefix of an index name. If a prefix is ambiguous, an error occurs.

Index hints do not work for FULLTEXT indexes.

USE INDEX, IGNORE INDEX, and FORCE INDEX affect only which indexes are used when MySQL decides how to find rows in the table and how to do the join. They do not affect whether an index is used when resolving an ORDER BY or GROUP BY clause.

Examples:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
  WHERE col1=1 AND col2=2 AND col3=3;

SELECT * FROM table1 IGNORE INDEX (col3_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

Index hints are accepted but ignored for `UPDATE` statements.

12.2.7.3 `UNION` Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

`UNION` is used to combine the result from multiple `SELECT` statements into a single result set. `UNION` is available from MySQL 4.0.0 on.

The column names from the first `SELECT` statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each `SELECT` statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

As of MySQL 4.1.1, if the data types of corresponding `SELECT` columns do not match, the types and lengths of the columns in the `UNION` result take into account the values retrieved by all of the `SELECT` statements. For example, consider the following:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
| bbbbbbbbbb    |
+---------------+
```

Before MySQL 4.1.1, only the type and length from the first `SELECT` would have been used and the second row would have been truncated to a length of 1:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
| b             |
+---------------+
```

The `SELECT` statements are normal select statements, but with the following restrictions:

- Only the last `SELECT` statement can use `INTO OUTFILE`. (However, the entire `UNION` result is written to the file.)
- `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`. If you specify it for the first `SELECT`, it has no effect. If you specify it for any subsequent `SELECT` statements, a syntax error results.

The default behavior for `UNION` is that duplicate rows are removed from the result. The optional `DISTINCT` keyword (introduced in MySQL 4.0.17) has no effect other than the default because it also specifies

duplicate-row removal. With the optional ALL keyword, duplicate-row removal does not occur and the result includes all matching rows from all the SELECT statements.

Before MySQL 4.1.2, you cannot mix UNION ALL and UNION DISTINCT in the same query. If you use ALL for one UNION, it is used for all of them. As of MySQL 4.1.2, mixed UNION types are treated such that a DISTINCT union overrides any ALL union to its left. A DISTINCT union can be produced explicitly by using UNION DISTINCT or implicitly by using UNION with no following DISTINCT or ALL keyword.

To use an ORDER BY or LIMIT clause to sort or limit the entire UNION result, parenthesize the individual SELECT statements and place the ORDER BY or LIMIT after the last one. The following example uses both clauses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

This kind of ORDER BY cannot use column references that include a table name (that is, names in *tbl_name*.*col_name* format). Instead, provide a column alias in the first SELECT statement and refer to the alias in the ORDER BY. (Alternatively, refer to the column in the ORDER BY using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the ORDER BY clause *must* refer to the alias, not the column name. The first of the following statements will work, but the second will fail with an Unknown column 'a' in 'order clause' error:

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To apply ORDER BY or LIMIT to an individual SELECT, place the clause inside the parentheses that enclose the SELECT:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

However, use of ORDER BY for individual SELECT statements implies nothing about the order in which the rows appear in the final result because UNION by default produces an unordered set of rows. Therefore, the use of ORDER BY in this context is typically in conjunction with LIMIT, so that it is used to determine the subset of the selected rows to retrieve for the SELECT, even though it does not necessarily affect the order of those rows in the final UNION result. If ORDER BY appears without LIMIT in a SELECT, it is optimized away because it will have no effect anyway.

To cause rows in a UNION result to consist of the sets of rows retrieved by each SELECT one after the other, select an additional column in each SELECT to use as a sort column and add an ORDER BY following the last SELECT:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual SELECT results, add a secondary column to the ORDER BY clause:

```
```

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

Use of an additional column also enables you to determine which `SELECT` each row comes from. Extra columns can provide other identifying information as well, such as a string that indicates a table name.

12.2.8 Subquery Syntax

A subquery is a `SELECT` statement within another statement.

Starting with MySQL 4.1, all subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

With MySQL versions prior to 4.1, it was necessary to work around or avoid the use of subqueries. In many cases, subqueries can successfully be rewritten using joins and other methods. See Section 12.2.8.11, “Rewriting Subqueries as Joins for Earlier MySQL Versions”.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- Many people find subqueries more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
 (SELECT COUNT(*) /* no hint */ FROM t2
  WHERE NOT EXISTS
   (SELECT * FROM t3
    WHERE ROW(5*t2.s1,77)=
     (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
      (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain many of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

One restriction is that a subquery's outer statement must be one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`. Another restriction is that currently you cannot modify a table and select from the same table in a subquery. This applies to statements such as `DELETE`, `INSERT`, `REPLACE`, and `UPDATE`.

A more comprehensive discussion of restrictions on subquery use, including performance issues for certain forms of subquery syntax, is given in Section D.1, “Restrictions on Subqueries”.

12.2.8.1 The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication that it can be `NULL`, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this `SELECT` returns a single value (`'abcde'`) that has a data type of `CHAR`, a length of 5, a character set and collation equal to the defaults in effect at `CREATE TABLE` time, and an indication that the value in the column can be `NULL`. Nullability of the value selected by a scalar subquery is not copied because if the subquery result is empty, the result is `NULL`. For the subquery just shown, if `t1` were empty, the result would be `NULL` even though `s2` is `NOT NULL`.

There are a few contexts in which a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, `LIMIT` requires literal integer arguments, and `LOAD DATA INFILE` requires a literal string file name. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct `(SELECT column1 FROM t1)`, imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Then perform a `SELECT`:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is `2` because there is a row in `t2` containing a column `s1` that has a value of `2`.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

12.2.8.2 Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *comparison_operator* is one of these operators:

```
=  >  <  >=  <=  <>  !=  <=>
```

For example:

```
... WHERE 'a' = (SELECT column1 FROM t1)
```

MySQL also permits this construct:

```
non_subquery_operand LIKE (subquery)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the rows in table `t1` for which the `column1` value is equal to a maximum value in table `t2`:

```
SELECT * FROM t1
  WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table `t1` containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
  WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison of the subquery to a scalar, the subquery must return a scalar. For a comparison of the subquery to a row constructor, the subquery must be a row subquery that returns a row with the same number of values as the row constructor. See Section 12.2.8.5, “Row Subqueries”.

12.2.8.3 Subqueries with ANY, IN, or SOME

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

Where *comparison_operator* is one of these operators:

```
=  >  <  >=  <=  <>  !=
```

The `ANY` keyword, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ANY` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(21,14,7)` because there is a value `7` in `t2` that is less than `10`. The expression is `FALSE` if table `t2` contains `(20,10)`, or if table `t2` is empty. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(NULL,NULL,NULL)`.

When used with a subquery, the word `IN` is an alias for `= ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

`IN` and `= ANY` are not synonyms when used with an expression list. `IN` can take an expression list, but `= ANY` cannot. See Section 11.3.2, “Comparison Functions and Operators”.

`NOT IN` is not an alias for `<> ANY`, but for `<> ALL`. See Section 12.2.8.4, “Subqueries with ALL”.

The word `SOME` is an alias for `ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word `SOME` is rare, but this example shows why it might be useful. To most people, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The syntax means “there is some b to which a is not equal.” Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

12.2.8.4 Subqueries with `ALL`

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word `ALL`, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ALL` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(-5,0,+5)` because `10` is greater than all three values in `t2`. The expression is `FALSE` if table `t2` contains `(12,6,NULL,-100)` because there is a single value `12` in table `t2` that is greater than `10`. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(0,NULL,1)`.

Finally, the expression is `TRUE` if table `t2` is empty. So, the following expression is `TRUE` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing `NULL` values* and *empty tables* are “edge cases.” When writing subqueries, always consider whether you have taken those two possibilities into account.

`NOT IN` is an alias for `<> ALL`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

12.2.8.5 Row Subqueries

The discussion to this point has been of scalar or column subqueries; that is, subqueries that return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Legal operators for row subquery comparisons are:

```
=  >  <  >=  <=  <>  !=  <=>
```

Here are two examples:

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

For both queries, if the table `t2` contains a single row with `id = 10`, the subquery returns a single row. If this row has `col3` and `col4` values equal to the `col1` and `col2` values of any rows in `t1`, the `WHERE` expression is `TRUE` and each query returns those `t1` rows. If the `t2` row `col3` and `col4` values are not equal the `col1` and `col2` values of any `t1` row, the expression is `FALSE` and the query returns an empty result set. The expression is *unknown* (that is, `NULL`) if the subquery produces no rows. An error occurs if the subquery produces multiple rows because a row subquery can return at most one row.

The expressions `(1,2)` and `ROW(1,2)` are sometimes called *row constructors*. The two are equivalent. The row constructor and the row returned by the subquery must contain the same number of values.

A row constructor is used for comparisons with subqueries that return two or more columns. When a subquery returns a single column, this is regarded as a scalar value and not as a row, so a row constructor cannot be used with a subquery that does not return at least two columns. Thus, the following query fails with a syntax error:

```
SELECT * FROM t1 WHERE ROW(1) = (SELECT column1 FROM t2)
```

Row constructors are legal in other contexts. For example, the following two statements are semantically equivalent (although in MySQL 4.1 only the second one can be optimized):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The following query answers the request, “find all rows in table `t1` that also exist in table `t2`”:

```
SELECT column1,column2,column3
  FROM t1
  WHERE (column1,column2,column3) IN
         (SELECT column1,column2,column3 FROM t2);
```

12.2.8.6 Subqueries with `EXISTS` or `NOT EXISTS`

If a subquery returns any rows at all, `EXISTS subquery` is `TRUE`, and `NOT EXISTS subquery` is `FALSE`. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an `EXISTS` subquery starts with `SELECT *`, but it could begin with `SELECT 5` or `SELECT column1` or anything at all. MySQL ignores the `SELECT` list in such a subquery, so it makes no difference.

For the preceding example, if `t2` contains any rows, even rows with nothing but `NULL` values, the `EXISTS` condition is `TRUE`. This is actually an unlikely example because a `[NOT] EXISTS` subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE EXISTS (SELECT * FROM cities_stores
                WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in no cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE NOT EXISTS (SELECT * FROM cities_stores
                    WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in all cities?

```
SELECT DISTINCT store_type FROM stores s1
  WHERE NOT EXISTS (
    SELECT * FROM cities WHERE NOT EXISTS (
      SELECT * FROM cities_stores
       WHERE cities_stores.city = cities.city
       AND cities_stores.store_type = stores.store_type));
```

The last example is a double-nested `NOT EXISTS` query. That is, it has a `NOT EXISTS` clause within a `NOT EXISTS` clause. Formally, it answers the question “does a city exist with a store that is not in `Stores`”? But it is easier to say that a nested `NOT EXISTS` answers the question “is *`x`* `TRUE` for all *`y`*?”

12.2.8.7 Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1
  WHERE column1 = ANY (SELECT column1 FROM t2
                       WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of `t1`, even though the subquery's `FROM` clause does not mention a table `t1`. So, MySQL looks outside the subquery, and finds `t1` in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5,6)` is not equal to `(5,7)`), so the expression as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
  WHERE x.column1 = (SELECT column1 FROM t2 AS x
    WHERE x.column1 = (SELECT column1 FROM t3
```

```
      WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Aggregate functions in correlated subqueries may contain outer references, provided the function contains nothing but outer references, and provided the function is not contained in another function or expression.

12.2.8.8 Subqueries in the `FROM` Clause

Subqueries are legal in a `SELECT` statement's `FROM` clause. The actual syntax is:

```
SELECT ... FROM (subquery) [AS] name ...
```

The `[AS] name` clause is mandatory, because every table in a `FROM` clause must have a name. Any columns in the *`subquery`* select list must have unique names.

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the `FROM` clause, using the example table:

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
  FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
  WHERE sb1 > 1;
```

Result: `2, '2', 4.0`.

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

However, this query provides the desired information:

```
SELECT AVG(sum_column1)
  FROM (SELECT SUM(column1) AS sum_column1
        FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (`sum_column1`) is recognized in the outer query.

Subqueries in the FROM clause can return a scalar, column, row, or table. Subqueries in the FROM clause cannot be correlated subqueries.

Subqueries in the FROM clause are executed even for the EXPLAIN statement (that is, derived temporary tables are built). This occurs because upper-level queries need information about all tables during the optimization phase, and the table represented by a subquery in the FROM clause is unavailable unless the subquery is executed.

It is possible under certain circumstances to modify table data using EXPLAIN SELECT. This can occur if the outer query accesses any tables and an inner query invokes a stored function that changes one or more rows of a table. Suppose that there are two tables t1 and t2 in database d1, created as shown here:

```
mysql> CREATE DATABASE d1;
Query OK, 1 row affected (0.00 sec)

mysql> USE d1;
Database changed

mysql> CREATE TABLE t1 (c1 INT);
Query OK, 0 rows affected (0.15 sec)

mysql> CREATE TABLE t2 (c1 INT);
Query OK, 0 rows affected (0.08 sec)
```

Now we create a stored function f1 which modifies t2:

```
mysql> DELIMITER //
mysql> CREATE FUNCTION f1(p1 INT) RETURNS INT
mysql>   BEGIN
mysql>     INSERT INTO t2 VALUES (p1);
mysql>     RETURN p1;
mysql>   END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

Referencing the function directly in an EXPLAIN SELECT does not have any effect on t2, as shown here:

```
mysql> SELECT * FROM t2;
Empty set (0.00 sec)

mysql> EXPLAIN SELECT f1(5);
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
|  1 | SIMPLE      | NULL  | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

This is because the SELECT statement did not reference any tables, as can be seen in the table and Extra columns of the output. This is also true of the following nested SELECT:

```
mysql> EXPLAIN SELECT NOW() AS a1, (SELECT f1(5)) AS a2;
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
|  1 | PRIMARY     | NULL  | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
```

```
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Note  | 1249 | Select 2 was reduced during optimization |
+-------+------+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

However, if the outer `SELECT` references any tables, the optimizer executes the statement in the subquery as well:

```
mysql> EXPLAIN SELECT * FROM t1 AS a1, (SELECT f1(5)) AS a2;
+----+-------------+------------+--------+---------------+------+---------+------+------+----------------------
| id | select_type | table      | type   | possible_keys | key  | key_len | ref  | rows | Extra
+----+-------------+------------+--------+---------------+------+---------+------+------+----------------------
|  1 | PRIMARY     | a1         | system | NULL          | NULL | NULL    | NULL |    0 | const row not found
|  1 | PRIMARY     | <derived2> | system | NULL          | NULL | NULL    | NULL |    1 |
|  2 | DERIVED     | NULL       | NULL   | NULL          | NULL | NULL    | NULL | NULL | No tables used
+----+-------------+------------+--------+---------------+------+---------+------+------+----------------------
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+
| c1   |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

This also means that an `EXPLAIN SELECT` statement such as the one shown here may take a long time to execute because the `BENCHMARK()` [870] function is executed once for each row in `t1`:

```
EXPLAIN SELECT * FROM t1 AS a1, (SELECT BENCHMARK(1000000, MD5(NOW())));
```

12.2.8.9 Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

  ```
  ERROR 1235 (ER_NOT_SUPPORTED_YET)
  SQLSTATE = 42000
  Message = "This version of MySQL doesn't yet support
  'LIMIT & IN/ALL/ANY/SOME subquery'"
  ```

  This means that MySQL does not support statements of the following form:

  ```
  SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
  ```

- Incorrect number of columns from subquery:

  ```
  ERROR 1241 (ER_OPERAND_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

This error occurs in cases like this:

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

You may use a subquery that returns multiple columns, if the purpose is row comparison. In other contexts, the subquery must be a scalar operand. See Section 12.2.8.5, “Row Subqueries”.

- Incorrect number of rows from subquery:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

This error occurs for statements where the subquery must return at most one row but returns multiple rows. Consider the following example:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

If `SELECT column1 FROM t2` returns just one row, the previous query will work. If the subquery returns more than one row, error 1242 will occur. In that case, the query should be rewritten as:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Incorrectly used table in subquery:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

This error occurs in cases such as the following, which attempts to modify a table and select from the same table in the subquery:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

You can use a subquery for assignment within an `UPDATE` statement because subqueries are legal in `UPDATE` and `DELETE` statements as well as in `SELECT` statements. However, you cannot use the same table (in this case, table `t1`) for both the subquery `FROM` clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For nontransactional storage engines, data modifications made before the error was encountered are preserved.

12.2.8.10 Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with:

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT DISTINCT column1 FROM t2);
```

```
SELECT * FROM t1 WHERE EXISTS
  (SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
  SELECT column1 FROM t2);
```

  Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2
  WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL before 4.1 that do not support subqueries. However, in some cases, converting a subquery to a join may also improve performance. See Section 12.2.8.11, “Rewriting Subqueries as Joins for Earlier MySQL Versions”.

- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

  Instead of this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

  For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

  Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

  Instead of this query:

```
SELECT * FROM t1
  WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
                AND t2.column2=t1.column2);
```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.

- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.

- Use `= ANY` rather than `EXISTS`.

- For uncorrelated subqueries that always return one row, `IN` is always slower than `=`. For example, use this query:

```
SELECT * FROM t1
  WHERE t1.col_name = (SELECT a FROM t2 WHERE b = some_const);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE t1.col_name IN (SELECT a FROM t2 WHERE b = some_const);
```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` [870] function, you can get an idea about what helps in your own situation. See Section 11.13, “Information Functions”.

Some optimizations that MySQL itself makes are:

- MySQL executes uncorrelated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is uncorrelated.
- MySQL rewrites `IN`, `ALL`, `ANY`, and `SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type (`unique_subquery` [568] or `index_subquery` [568]):

  ```
  ... IN (SELECT indexed_column FROM single_table ...)
  ```

- MySQL enhances expressions of the following form with an expression involving `MIN()` [884] or `MAX()` [884], unless `NULL` values or empty sets are involved:

  ```
  value {ALL|ANY|SOME} {> | < | >= | <=} (uncorrelated subquery)
  ```

  For example, this `WHERE` clause:

  ```
  WHERE 5 > ALL (SELECT x FROM t)
  ```

  might be treated by the optimizer like this:

  ```
  WHERE 5 > (SELECT MAX(x) FROM t)
  ```

See also MySQL Internals: How MySQL Transforms Subqueries.

12.2.8.11 Rewriting Subqueries as Joins for Earlier MySQL Versions

Before MySQL 4.1, only nested queries of the form `INSERT ... SELECT ...` and `REPLACE ... SELECT ...` are supported. The `IN()` construct can be used in other contexts to test membership in a set of values.

It is often possible to rewrite a query without a subquery:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

This can be rewritten as:

```
```

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.*
  FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better—a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

For more complicated subqueries, you can often create temporary tables to hold the subquery. In some cases, however, this option does not work. The most frequently encountered of these cases arises with `DELETE` statements, for which standard SQL does not support joins (except in subqueries). For this situation, there are three options available:

- The first option is to upgrade to MySQL 4.1, which does support subqueries in `DELETE` statements.
- The second option is to use a procedural programming language (such as Perl or PHP) to submit a `SELECT` query which obtains the primary keys for the rows to be deleted, and then use these values to construct the appropriate `DELETE` statement (`DELETE FROM ... WHERE key_col IN (key1, key2,...)`).
- The third option is to use interactive SQL to construct a set of `DELETE` statements automatically, using the MySQL extension `CONCAT()` [795] (in lieu of the standard `||` [787] operator). For example:

  ```
  SELECT
    CONCAT('DELETE FROM tab1 WHERE pkid = ', "'", tab1.pkid, "'", ';')
    FROM tab1, tab2
   WHERE tab1.col1 = tab2.col2;
  ```

  You can place this query in a script file, use the file as input to one instance of the `mysql` program, and use the program output as input to a second instance of `mysql`:

  ```
  shell> mysql --skip-column-names mydb < myscript.sql | mysql mydb
  ```

MySQL Server 4.0 supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported as of MySQL 4.0.

12.2.9 UPDATE Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
```

For the single-table syntax, the UPDATE statement updates columns of existing rows in the named table with new values. The SET clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword DEFAULT to set a column explicitly to its default value. The WHERE clause, if given, specifies the conditions that identify which rows to update. With no WHERE clause, all rows are updated. If the ORDER BY clause is specified, the rows are updated in the order that is specified. The LIMIT clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, UPDATE updates rows in each table named in *table_references* that satisfy the conditions. Each matching row is updated once, even if it matches the conditions multiple times. For multiple-table syntax, ORDER BY and LIMIT cannot be used.

where_condition is an expression that evaluates to true for each row to be updated. For expression syntax, see Section 8.5, “Expression Syntax”.

table_references and *where_condition* are is specified as described in Section 12.2.7, “SELECT Syntax”.

You need the UPDATE [493] privilege only for columns referenced in an UPDATE that are actually updated. You need only the SELECT [492] privilege for any columns that are read but not modified.

The UPDATE statement supports the following modifiers:

- With the LOW_PRIORITY keyword, execution of the UPDATE is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).
- With the IGNORE keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closest valid values instead.

If you access a column from the table to be updated in an expression, UPDATE uses the current value of the column. For example, the following statement sets col1 to one more than its current value:

```
UPDATE t1 SET col1 = col1 + 1;
```

The second assignment in the following statement sets col2 to the current (updated) col1 value, not the original col1 value. The result is that col1 and col2 have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

Single-table UPDATE assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared NOT NULL by setting to NULL, the column is set to the default value appropriate for the data type and the warning count is incremented. The default value is 0 for numeric types, the empty string ('') for string types, and the “zero” value for date and time types.

`UPDATE` returns the number of rows that were actually changed. In MySQL 3.22 or later, the `mysql_info()` C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the `UPDATE`.

You can use `LIMIT row_count` to restrict the scope of the `UPDATE`. A `LIMIT` clause works as follows:

- Before MySQL 4.0.13, `LIMIT` is a rows-affected restriction. The statement stops as soon as it has changed `row_count` rows that satisfy the `WHERE` clause.
- From 4.0.13 on, `LIMIT` is a rows-matched restriction. The statement stops as soon as it has found `row_count` rows that satisfy the `WHERE` clause, whether or not they actually were changed.

If an `UPDATE` statement includes an `ORDER BY` clause, the rows are updated in the order specified by the clause. `ORDER BY` can be used from MySQL 4.0.0. This can be useful in certain situations that might otherwise result in an error. Suppose that a table `t` contains a column `id` that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the `id` column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an `ORDER BY` clause to cause the rows with larger `id` values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

Starting with MySQL 4.0.4, you can also perform `UPDATE` operations covering multiple tables. However, you cannot use `ORDER BY` or `LIMIT` with a multiple-table `UPDATE`. The `table_references` clause lists the tables involved in the join. Its syntax is described in Section 12.2.7.1, “`JOIN` Syntax”. Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table `UPDATE` statements can use any type of join permitted in `SELECT` statements, such as `LEFT JOIN`.

Before MySQL 4.0.18, you need the `UPDATE` [493] privilege for all tables used in a multiple-table `UPDATE`, even if they were not updated. As of MySQL 4.0.18, you need only the `SELECT` [492] privilege for any columns that are read but not modified.

If you use a multiple-table `UPDATE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the `ON UPDATE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly. See Section 13.2.5.4, “`FOREIGN KEY` Constraints”.

Currently, you cannot update a table and select from the same table in a subquery.

Index hints (see Section 12.2.7.2, “Index Hint Syntax”) are accepted but ignored for `UPDATE` statements.

12.3 MySQL Transactional and Locking Statements

MySQL supports local transactions (within a given client session) through statements such as `SET autocommit`, `START TRANSACTION`, `COMMIT`, and `ROLLBACK`. See Section 12.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”.

12.3.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax

```
START TRANSACTION [WITH CONSISTENT SNAPSHOT] | BEGIN [WORK]
COMMIT
ROLLBACK
SET autocommit = {0 | 1}
```

The START TRANSACTION or BEGIN statement begins a new transaction. COMMIT commits the current transaction, making its changes permanent. ROLLBACK rolls back the current transaction, canceling its changes. The SET autocommit statement disables or enables the default autocommit mode for the current session.

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent. To disable autocommit mode, use the following statement:

If you are using a transaction-safe storage engine (such as InnoDB, BDB, or NDBCLUSTER), you can disable autocommit mode with the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the autocommit [406] variable to zero, changes to transaction-safe tables (such as those for InnoDB or NDBCLUSTER) are not made permanent immediately. You must use COMMIT to store your changes to disk or ROLLBACK to ignore the changes.

autocommit [406] is a session variable and must be set for each session. For information about how to do this for each new connection, see the description of the init_connect [414] system variable.

To disable autocommit mode for a single series of statements, use the START TRANSACTION statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With START TRANSACTION, autocommit remains disabled until you end the transaction with COMMIT or ROLLBACK. The autocommit mode then reverts to its previous state.

BEGIN and BEGIN WORK are supported as aliases of START TRANSACTION for initiating a transaction. START TRANSACTION was added in MySQL 4.0.11. This is standard SQL syntax and is the recommended way to start an ad-hoc transaction. BEGIN and BEGIN WORK are available from MySQL 3.23.17 and 3.23.19, respectively.

Important

Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a START TRANSACTION statement from the client. See Chapter 17, *Connectors and APIs*, or the documentation for your API, for more information.

As of MySQL 4.1.8, you can begin a transaction like this:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

The `WITH CONSISTENT SNAPSHOT` clause starts a consistent read for storage engines that are capable of it. This applies only to `InnoDB`. The effect is the same as issuing a `START TRANSACTION` followed by a `SELECT` from any `InnoDB` table. See Section 13.2.9.2, “Consistent Nonlocking Reads”. The `WITH CONSISTENT SNAPSHOT` clause does not change the current transaction isolation level, so it provides a consistent snapshot only if the current isolation level is one that permits consistent read (`REPEATABLE READ` [976] or `SERIALIZABLE` [976]).

Beginning a transaction causes any pending transaction to be committed. See Section 12.3.3, “Statements That Cause an Implicit Commit”, for more information.

Beginning a transaction also causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

For best results, transactions should be performed using only tables managed by a single transaction-safe storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as `InnoDB` and `BDB`), and the transaction isolation level is not `SERIALIZABLE` [976], it is possible that when one transaction commits, another ongoing transaction that uses the same tables will see only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use `SET TRANSACTION ISOLATION LEVEL` to set the isolation level to `SERIALIZABLE` [976] on a per-transaction basis as necessary.)
- If you use tables that are not transaction-safe within a transaction, changes to those tables are stored at once, regardless of the status of autocommit mode.
- If you issue a `ROLLBACK` statement after updating a nontransactional table within a transaction, an `ER_WARNING_NOT_COMPLETE_ROLLBACK` [1580] warning occurs. Changes to transaction-safe tables are rolled back, but not changes to nontransaction-safe tables.

If you are using `START TRANSACTION` or `SET autocommit = 0`, you should use the MySQL binary log for backups instead of the older update log. Transactions are stored in the binary log in one chunk, upon `COMMIT`. Transactions that are rolled back are not logged. (**Exception**: Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that modifications to the nontransactional tables are replicated. This is true as of MySQL 4.0.15.) See Section 5.3.4, “The Binary Log”.

You can change the isolation level for transactions with `SET TRANSACTION ISOLATION LEVEL`. See Section 12.3.6, “`SET TRANSACTION` Syntax”.

Rolling back can be a slow operation that may occur implicitly without the user having explicitly asked for it (for example, when an error occurs). Because of this, as of MySQL 4.1.8, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the session, not only for explicit rollbacks performed with the `ROLLBACK` statement but also for implicit rollbacks.

12.3.2 Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases or those that create, drop, or alter tables.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a `ROLLBACK` statement.

12.3.3 Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end a transaction, as if you had done a `COMMIT` before executing the statement.

- **Data definition language (DDL) statements that define or modify database objects.** `ALTER TABLE`, `CREATE INDEX`, `DROP INDEX`, `DROP TABLE`, `RENAME TABLE`.

  `ALTER TABLE`, `CREATE TABLE`, and `DROP TABLE` do not commit a transaction if the `TEMPORARY` keyword is used. (This does not apply to other operations on temporary tables such as `CREATE INDEX`, which do cause a commit.) However, although no implicit commit occurs, neither can the statement be rolled back. Therefore, use of such statements will violate transaction atomicity: For example, if you use `CREATE TEMPORARY TABLE` and then roll back the transaction, the table remains in existence.

  The `CREATE TABLE` statement in `InnoDB` is processed as a single transaction. This means that a `ROLLBACK` from the user does not undo `CREATE TABLE` statements the user made during that transaction.

  Prior to MySQL 4.0.13, `CREATE TABLE` commits a transaction if the binary update log is enabled. The `CREATE TABLE`, `CREATE DATABASE` `DROP DATABASE`, and `TRUNCATE TABLE` statements cause an implicit commit beginning with MySQL 4.1.13.

- **Transaction-control and locking statements.** `BEGIN`, `LOCK TABLES`, `SET autocommit = 1` (if the value is not already 1), `START TRANSACTION`, `UNLOCK TABLES`.

  `UNLOCK TABLES` commits a transaction only if any tables currently have been locked with `LOCK TABLES`. This does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table-level locks.

  Transactions cannot be nested. This is a consequence of the implicit commit performed for any current transaction when you issue a `START TRANSACTION` statement or one of its synonyms.

- **Data loading statements.** `LOAD MASTER DATA`.

12.3.4 `SAVEPOINT` and `ROLLBACK TO SAVEPOINT` Syntax

```
SAVEPOINT identifier
ROLLBACK TO SAVEPOINT identifier
```

Starting from MySQL 4.0.14 and 4.1.1, `InnoDB` supports the SQL statements `SAVEPOINT` and `ROLLBACK TO SAVEPOINT`.

The `SAVEPOINT` statement sets a named transaction savepoint with a name of *`identifier`*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The `ROLLBACK TO SAVEPOINT` statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but `InnoDB` does *not* release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the `ROLLBACK TO SAVEPOINT` statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1181: Got error 153 during ROLLBACK
```

All savepoints of the current transaction are deleted if you execute a COMMIT, or a ROLLBACK that does not name a savepoint.

12.3.5 LOCK TABLES and UNLOCK TABLES Syntax

```
LOCK TABLES
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type:
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE

UNLOCK TABLES
```

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail later in this section.

LOCK TABLES acquires table locks for the current client session. As of MySQL 4.0.2, to use LOCK TABLES you must have the LOCK TABLES [492] privilege, and the SELECT [492] privilege for each table to be locked. In MySQL 3.23, you must have SELECT [492], INSERT [492], DELETE [491], and UPDATE [493] privileges for the tables.

UNLOCK TABLES explicitly releases any table locks held by the current session.

Another use for UNLOCK TABLES is to release the global read lock acquired with the FLUSH TABLES WITH READ LOCK statement, which enables you to lock all tables in all databases. See Section 12.4.6.2, “FLUSH Syntax”. (This is a very convenient way to get backups if you have a file system such as Veritas that can take snapshots in time.)

A table lock protects only against inappropriate reads or writes by other sessions. The session holding the lock, even a read lock, can perform table-level operations such as DROP TABLE. Truncate operations are not transaction-safe, so an error occurs if the session attempts one during an active transaction or while holding a table lock.

The following discussion applies only to non-TEMPORARY tables. LOCK TABLES is permitted (but ignored) for a TEMPORARY table. The table can be accessed freely by the session within which it was created, regardless of what other locking may be in effect. No lock is necessary because no other session can see the table.

For information about other conditions on the use of LOCK TABLES and statements that cannot be used while LOCK TABLES is in effect, see Section 12.3.5.2, “Table-Locking Restrictions and Conditions”

Rules for Lock Acquisition

To acquire table locks within the current session, use the LOCK TABLES statement. The following lock types are available:

READ [LOCAL] lock:

- The session that holds the lock can read the table (but not write it).
- Multiple sessions can acquire a READ lock for the table at the same time.
- Other sessions can read the table without explicitly acquiring a READ lock.
- The LOCAL modifier enables nonconflicting INSERT statements (concurrent inserts) by other sessions to execute while the lock is held. (See Section 7.6.3, "Concurrent Inserts".) However, READ LOCAL cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For InnoDB tables, READ LOCAL is the same as READ as of MySQL 4.1.15. (Before that, READ LOCAL essentially does nothing: It does not lock the table at all, so for InnoDB tables, the use of READ LOCAL is deprecated because a plain consistent-read SELECT does the same thing, and no locks are needed.)

[LOW_PRIORITY] WRITE lock:

- The session that holds the lock can read and write the table.
- Only the session that holds the lock can access the table. No other session can access it until the lock is released.
- Lock requests for the table by other sessions block while the WRITE lock is held.
- The LOW_PRIORITY modifier affects lock scheduling if the WRITE lock request must wait, as described later.

If the LOCK TABLES statement must wait due to locks held by other sessions on any of the tables, it blocks until all locks can be acquired.

A session that requires locks must acquire all the locks that it needs in a single LOCK TABLES statement. While the locks thus obtained are held, the session can access only the locked tables. For example, in the following sequence of statements, an error occurs for the attempt to access t2 because it was not locked in the LOCK TABLES statement:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

You cannot refer to a locked table multiple times in a single query using the same name. Use aliases instead, and obtain a separate lock for the table and each alias:

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

The error occurs for the first INSERT because there are two references to the same name for a locked table. The second INSERT succeeds because the references to the table use different names.

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

`WRITE` locks normally have higher priority than `READ` locks to ensure that updates are processed as soon as possible. This means that if one session obtains a `READ` lock and then another session requests a `WRITE` lock, subsequent `READ` lock requests wait until the session that requested the `WRITE` lock has obtained the lock and released it. A request for a `LOW_PRIORITY WRITE` lock, by contrast, permits subsequent `READ` lock requests by other sessions to be satisfied first if they occur while the `LOW_PRIORITY WRITE` request is waiting. You should use `LOW_PRIORITY WRITE` locks only if you are sure that eventually there will be a time when no sessions have a `READ` lock. For `InnoDB` tables in transactional mode (autocommit = 0), a waiting `LOW_PRIORITY WRITE` lock acts like a regular `WRITE` lock and causes subsequent `READ` lock requests to wait.

`LOCK TABLES` acquires locks as follows:

1. Sort all tables to be locked in an internally defined order. From the user standpoint, this order is undefined.
2. If a table is to be locked with a read and a write lock, put the write lock request before the read lock request.
3. Lock one table at a time until the session gets all locks.

This policy ensures that table locking is deadlock free. There are, however, other things you need to be aware of about this policy: If you are using a `LOW_PRIORITY WRITE` lock for a table, it means only that MySQL waits for this particular lock until there are no other sessions that want a `READ` lock. When the session has gotten the `WRITE` lock and is waiting to get the lock for the next table in the lock table list, all other sessions wait for the `WRITE` lock to be released. If this becomes a serious problem with your application, you should consider converting some of your tables to transaction-safe tables.

Rules for Lock Release

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly, or locks may be released implicitly under certain conditions.

- A session can release its locks explicitly with `UNLOCK TABLES`.
- If a session issues a `LOCK TABLES` statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are granted.
- If a session begins a transaction (for example, with `START TRANSACTION`), an implicit `UNLOCK TABLES` is performed, which causes existing locks to be released. (For additional information about the interaction between table locking and transactions, see Section 12.3.5.1, “Interaction of Table Locking and Transactions”.)

If the connection for a client session terminates, whether normally or abnormally, the server implicitly releases all table locks held by the session (transactional and nontransactional). If the client reconnects, the locks will no longer be in effect. In addition, if the client had an active transaction, the server rolls

back the transaction upon disconnect, and if reconnect occurs, the new session begins with autocommit enabled. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks or current transaction will have been lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction. See Section 17.6.14, “Controlling Automatic Reconnection Behavior”.

Note

If you use ALTER TABLE on a locked table, it may become unlocked. For example, if you attempt a second ALTER TABLE operation, the result may be an error Table '*tbl_name*' was not locked with LOCK TABLES. To handle this, lock the table again prior to the second alteration. See also Section B.5.7.1, “Problems with ALTER TABLE”.

12.3.5.1 Interaction of Table Locking and Transactions

LOCK TABLES and UNLOCK TABLES interact with the use of transactions as follows:

- LOCK TABLES is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables.
- UNLOCK TABLES implicitly commits any active transaction, but only if LOCK TABLES has been used to acquire table locks. For example, in the following set of statements, UNLOCK TABLES releases the global read lock but does not commit the transaction because no table locks are in effect:

```
FLUSH TABLES WITH READ LOCK;
START TRANSACTION;
SELECT ... ;
UNLOCK TABLES;
```

- Beginning a transaction (for example, with START TRANSACTION) implicitly commits any current transaction and releases existing locks.
- Other statements that implicitly cause transactions to be committed do not release existing locks. For a list of such statements, see Section 12.3.3, “Statements That Cause an Implicit Commit”.
- The correct way to use LOCK TABLES and UNLOCK TABLES with transactional tables, such as InnoDB tables, is to begin a transaction with SET autocommit = 0 (not START TRANSACTION) followed by LOCK TABLES, and to not call UNLOCK TABLES until you commit the transaction explicitly. For example, if you need to write to table t1 and read from table t2, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

  When you call LOCK TABLES, InnoDB internally takes its own table lock, and MySQL takes its own table lock. InnoDB releases its internal table lock at the next commit, but for MySQL to release its table lock, you have to call UNLOCK TABLES. You should not have autocommit = 1 [406], because then InnoDB releases its internal table lock immediately after the call of LOCK TABLES, and deadlocks can very easily happen. Starting from 4.1.9, InnoDB does not acquire the internal table lock at all if autocommit = 1 [406], to help old applications avoid unnecessary deadlocks.

- ROLLBACK does not release table locks.

- FLUSH TABLES WITH READ LOCK acquires a global read lock and not table locks, so it is not subject to the same behavior as LOCK TABLES and UNLOCK TABLES with respect to table locking and implicit commits. See Section 12.4.6.2, "FLUSH Syntax".

12.3.5.2 Table-Locking Restrictions and Conditions

You can safely use KILL to terminate a session that is waiting for a table lock. See Section 12.4.6.3, "KILL Syntax".

You should *not* lock any tables that you are using with INSERT DELAYED. An INSERT DELAYED in this case results in an error because the insert must be handled by a separate thread, not by the session which holds the lock.

Normally, you do not need to lock tables, because all single UPDATE statements are atomic; no other session can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of MyISAM tables, it is much faster to lock the tables you are going to use. Locking MyISAM tables speeds up inserting, updating, or deleting on them because MySQL does not flush the key cache for the locked tables until UNLOCK TABLES is called. Normally, the key cache is flushed after each SQL statement.

  The downside to locking the tables is that no session can update a READ-locked table (including the one holding the lock) and no session can access a WRITE-locked table other than the one holding the lock.

- If you are using tables for a nontransactional storage engine, you must use LOCK TABLES if you want to ensure that no other session modifies the tables between a SELECT and an UPDATE. The example shown here requires LOCK TABLES to execute safely:

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

  Without LOCK TABLES, it is possible that another session might insert a new row in the trans table between execution of the SELECT and UPDATE statements.

You can avoid using LOCK TABLES in many cases by using relative updates (UPDATE customer SET *value*=*value*+*new_value*) or the LAST_INSERT_ID() [874] function. See Section 1.9.5.4, "Transactions and Atomic Operations".

You can also avoid locking tables in some cases by using the user-level advisory lock functions GET_LOCK() [877] and RELEASE_LOCK() [879]. These locks are saved in a hash table in the server and implemented with pthread_mutex_lock() and pthread_mutex_unlock() for high speed. See Section 11.14, "Miscellaneous Functions".

See Section 7.6.1, "Internal Locking Methods", for more information on locking policy.

12.3.6 SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
  {
       READ UNCOMMITTED
     | READ COMMITTED
     | REPEATABLE READ
```

```
    | SERIALIZABLE
}
```

This statement sets the transaction isolation level globally, for the current session, or for the next transaction:

- With the GLOBAL keyword, the statement sets the default transaction level globally for all subsequent sessions. Existing sessions are unaffected.
- With the SESSION keyword, the statement sets the default transaction level for all subsequent transactions performed within the current session.
- Without any SESSION or GLOBAL keyword, the statement sets the isolation level for the next (not started) transaction performed within the current session.

A change to the global default isolation level requires the SUPER [493] privilege. Any session is free to change its session isolation level (even in the middle of a transaction), or the isolation level for its next transaction.

To set the global default isolation level at server startup, use the `--transaction-isolation=level` [394] option to mysqld on the command line or in an option file. Values of *level* for this option use dashes rather than spaces, so the permissible values are READ-UNCOMMITTED [975], READ-COMMITTED [975], REPEATABLE-READ [976], or SERIALIZABLE [976]. For example, to set the default isolation level to REPEATABLE READ [976], use these lines in the [mysqld] section of an option file:

```
[mysqld]
transaction-isolation = REPEATABLE-READ
```

To determine the global and session transaction isolation levels at runtime, check the value of the tx_isolation [433] system variable:

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
```

As of MySQL 4.0.5, InnoDB supports each of the transaction isolation levels described here using different locking strategies. (Before 4.0.5, only REPEATABLE READ [976] and SERIALIZABLE [976] were available. Before MySQL 3.23.50, SET TRANSACTION had no effect on InnoDB tables.) The default level is REPEATABLE READ [976]. For additional information about InnoDB record-level locks and how it uses them to execute various types of statements, see Section 13.2.9.4, “InnoDB Record, Gap, and Next-Key Locks”, and Section 13.2.9.6, “Locks Set by Different SQL Statements in InnoDB”.

The following list describes how MySQL supports the different transaction levels:

- READ UNCOMMITTED [975]

  SELECT statements are performed in a nonlocking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a “dirty read.” Otherwise, this isolation level works like READ COMMITTED [975].

- READ COMMITTED [975]

  A somewhat Oracle-like isolation level with respect to consistent (nonlocking) reads: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See Section 13.2.9.2, “Consistent Nonlocking Reads”.

  For locking reads (SELECT with FOR UPDATE or LOCK IN SHARE MODE), InnoDB locks only index records, not the gaps before them, and thus permits the free insertion of new records next to locked

records. For `UPDATE` and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition (such as `WHERE id = 100`), or a range-type search condition (such as `WHERE id > 100`). For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it. For range-type searches, `InnoDB` locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range. This is necessary because “phantom rows” must be blocked for MySQL replication and recovery to work.

- `REPEATABLE READ` [976]

  This is the default isolation level for `InnoDB`. For consistent reads, there is an important difference from the `READ COMMITTED` [975] isolation level: All consistent reads within the same transaction read the snapshot established by the first read. This convention means that if you issue several plain (nonlocking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See Section 13.2.9.2, “Consistent Nonlocking Reads”.

  For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE`, and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition. For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it. For other search conditions, `InnoDB` locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range.

- `SERIALIZABLE` [976]

  This level is like `REPEATABLE READ` [976], but `InnoDB` implicitly converts all plain `SELECT` statements to `SELECT ... LOCK IN SHARE MODE` if autocommit is disabled. If autocommit is enabled, the `SELECT` is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (nonlocking) read and need not block for other transactions. (This means that to force a plain `SELECT` to block if other transactions have modified the selected rows, you should disable autocommit.)

12.4 Database Administration Statements

12.4.1 Account Management Statements

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in Chapter 5, *MySQL Server Administration*, which you should consult for additional details.

Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

12.4.1.1 `DROP USER` Syntax

```
DROP USER user [, user] ...
```

The `DROP USER` statement removes one or more MySQL accounts. To use it, you must have the `DELETE` [491] privilege for the `mysql` database. Each account is named using the same format as for the `GRANT` statement; for example, `'jeffrey'@'localhost'`. If you specify only the user name part of

the account name, a host name part of `'%'` is used. For additional information about specifying account names, see Section 12.4.1.2, “GRANT Syntax”.

`DROP USER` was added in MySQL 4.1.1. In MySQL 4.1, it serves only to remove account rows from the `user` table for accounts that have no privileges. To remove a MySQL account completely (including all of its privileges), you should use the following procedure, performing the steps in the order shown:

1. Use `SHOW GRANTS` to determine what privileges the account has. See Section 12.4.5.12, “SHOW GRANTS Syntax”.
2. Use `REVOKE` to revoke the privileges displayed by `SHOW GRANTS`. This removes rows for the account from all the grant tables except the `user` table, and revokes any global privileges listed in the `user` table. See Section 12.4.1.2, “GRANT Syntax”.
3. Delete the account by using `DROP USER` to remove the `user` table row.

In MySQL 5.0.2 and up, `DROP USER` removes the account row in the `user` table and also revokes the privileges held by the account. It is not necessary to use `DROP USER` in conjunction with `REVOKE`.

Important

`DROP USER` does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in will fail. *This is by design*.

Before MySQL 4.1.1, `DROP USER` is not available. You should first revoke the account privileges using `SHOW GRANTS` and `REVOKE` as just described. Then delete the `user` table row and flush the grant tables as shown here:

```
mysql> DELETE FROM mysql.user
    -> WHERE User='user_name' and Host='host_name';
mysql> FLUSH PRIVILEGES;
```

12.4.1.2 GRANT Syntax

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON priv_level
    TO user_specification [, user_specification] ...
    [REQUIRE {NONE | ssl_option [[AND] ssl_option] ...}]
    [WITH with_option ...]

priv_level:
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name

user_specification:
    user [IDENTIFIED BY [PASSWORD] 'password']

ssl_option:
    SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'
```

```
with_option =
    GRANT OPTION
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
```

The GRANT statement creates MySQL user accounts and grants rights to accounts. To use GRANT, you must have the GRANT OPTION [491] privilege, and you must have the privileges that you are granting. GRANT is implemented in MySQL 3.22.11 or later. For earlier MySQL versions, it does nothing.

The REVOKE statement is related to GRANT and enables administrators to remove account privileges. See Section 12.4.1.3, "REVOKE Syntax".

To determine what privileges an account has, use SHOW GRANTS. See Section 12.4.5.12, "SHOW GRANTS Syntax".

There are several aspects to the GRANT statement, described under the folllowing topics in this section:

- Privileges Supported by MySQL
- Global Privileges
- Database Privileges
- Table Privileges
- Column Privileges
- Account Names and Passwords
- Other Account Characteristics
- MySQL and Standard SQL Versions of GRANT

Important

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See Section 4.4.5, "mysql_fix_privilege_tables — Upgrade MySQL System Tables".

Privileges Supported by MySQL

The following table summarizes the permissible *priv_type* privilege types that can be specified for the GRANT and REVOKE statements. For additional information about these privileges, see Section 5.5.1, "Privileges Provided by MySQL".

Table 12.1 Permissible Privileges for GRANT and REVOKE

Privilege	Meaning
ALL [PRIVILEGES] [491]	Grant all privileges at specified access level except GRANT OPTION [491]
ALTER [491]	Enable use of ALTER TABLE
CREATE [491]	Enable database and table creation
CREATE TEMPORARY TABLES [491]	Enable use of CREATE TEMPORARY TABLE
DELETE [491]	Enable use of DELETE

Privilege	Meaning
DROP [491]	Enable databases, tables, and views to be dropped
EXECUTE [491]	Not implemented
FILE [491]	Enable the user to cause the server to read or write files
GRANT OPTION [491]	Enable privileges to be granted to or removed from other accounts
INDEX [491]	Enable indexes to be created or dropped
INSERT [492]	Enable use of INSERT
LOCK TABLES [492]	Enable use of LOCK TABLES on tables for which you have the SELECT privilege
PROCESS [492]	Enable the user to see all processes with SHOW PROCESSLIST
REFERENCES [492]	Not implemented
RELOAD [492]	Enable use of FLUSH operations
REPLICATION CLIENT [492]	Enable the user to ask where master or slave servers are
REPLICATION SLAVE [492]	Enable replication slaves to read binary log events from the master
SELECT [492]	Enable use of SELECT
SHOW DATABASES [492]	Enable SHOW DATABASES to show all databases
SHUTDOWN [492]	Enable use of mysqladmin shutdown
SUPER [493]	Enable use of other adminstrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin debug command
UPDATE [493]	Enable use of UPDATE
USAGE [493]	Synonym for “no privileges”

The CREATE TEMPORARY TABLES [491], EXECUTE [491], LOCK TABLES [492], REPLICATION CLIENT [492], REPLICATION SLAVE [492], SHOW DATABASES [492], and SUPER [493] privileges were added in MySQL 4.0.2. To use these privileges when upgrading from an earlier version of MySQL that does not have them, you must first upgrade the grant tables. See Section 4.4.5, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”.

The REFERENCES [492] and EXECUTE [491] privileges are unused in MySQL versions up to and including the 4.1 release series.

In older MySQL versions that do not have the SUPER [493] privilege, specify the PROCESS [492] privilege instead.

In GRANT statements, the ALL [PRIVILEGES] [491] privilege is named by itself and cannot be specified along with other privileges. It stands for all privileges available for the level at which privileges are to be granted except for the GRANT OPTION [491] privilege.

USAGE [493] can be specified to create a user that has no privileges, or to specify the REQUIRE or WITH clauses for an account without changing its existing privileges.

MySQL account information is stored in the tables of the mysql database. This database and the access control system are discussed extensively in Section 5.5, “The MySQL Access Privilege System”, which you should consult for additional details.

If the grant tables hold privilege rows that contain mixed-case database or table names and the lower_case_table_names [418] system variable is set to a nonzero value, REVOKE cannot be used to

revoke these privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create such rows when `lower_case_table_names` [418] is set, but such rows might have been created prior to setting that variable.)

Privileges can be granted at several levels, depending on the syntax used for the `ON` clause. For `REVOKE`, the same `ON` syntax specifies which privileges to take away. The examples shown here include no `IDENTIFIED BY 'password'` clause for brevity, but you should include one if the account does not already exist, to avoid creating an insecure account that has no password.

Global Privileges

Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use `ON *.*` syntax:

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

Privileges also are assigned at the global level if you use `ON *` syntax and you have *not* selected a default database.

The `FILE` [491], `PROCESS` [492], `RELOAD` [492], `REPLICATION CLIENT` [492], `REPLICATION SLAVE` [492], `SHOW DATABASES` [492], `SHUTDOWN` [492], and `SUPER` [493] privileges are administrative and can only be granted globally.

Other privileges can be granted globally or at more specific levels.

MySQL stores global privileges in the `mysql.user` table.

Database Privileges

Database privileges apply to all tables in a given database. To assign database-level privileges, use `ON db_name.*` syntax:

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

If you use `ON *` syntax (rather than `ON *.*` and you have selected a default database, privileges are assigned at the database level for the default database.

The `CREATE`, `DROP`, and `GRANT OPTION` [491] privileges can be specified at the database level. Table privileges also can be specified at the database level, in which case they apply to all tables in the database.

MySQL stores database privileges in the `mysql.db` table.

Table Privileges

Table privileges apply to all columns in a given table. To assign table-level privileges, use `ON db_name.tbl_name` syntax:

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

If you specify *`tbl_name`* rather than *`db_name.tbl_name`*, the statement applies to *`tbl_name`* in the default database. An error occurs if there is no default database.

The permissible *priv_type* values for a table are ALTER [491], CREATE [491], DELETE [491], DROP [491], GRANT OPTION [491], INDEX [491], INSERT [492], SELECT [492], and UPDATE [493].

MySQL stores table privileges in the mysql.tables_priv table.

Column Privileges

Column privileges apply to single columns in a given table. Each privilege to be granted at the column level must be followed by the column or columns, enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

The permissible column-level *priv_type* values are INSERT [492], SELECT [492], and UPDATE [493].

MySQL stores column privileges in the mysql.columns_priv table.

For the global, database, and table levels, GRANT ALL assigns only the privileges that exist at the level you are granting. For example, if you use GRANT ALL ON *db_name*.*, that is a database-level statement, so none of the global-only privileges such as FILE [491] are granted.

The privileges for a database, table, or column are formed additively as the logical OR [787] of the privileges at each of the privilege levels. For example, if a user has a global SELECT [492] privilege, the privilege cannot be denied by an absence of the privilege at the database, table, or column level. Details of the privilege-checking procedure are presented in Section 5.5.5, “Access Control, Stage 2: Request Verification”.

MySQL enables you to grant privileges even on databases and tables that do not exist. In such cases, the privileges to be granted must include the CREATE [491] privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for databases and tables that are to be created at a later time.

Important

MySQL does not automatically revoke any privileges when you drop a database or table.

Account Names and Passwords

The *user* value indicates the MySQL account to which the GRANT statement applies. To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *user* value in the form *user_name*@*host_name*. If a *user_name* or *host_name* value is legal as an unquoted identifier, you need not quote it. However, quotation marks are necessary to specify a *user_name* string containing special characters (such as “-”), or a *host_name* string containing special characters or wildcard characters (such as “%”); for example, 'test-user'@'%.com'. Quote the user name and host name separately.

You can specify wildcards in the host name. For example, *user_name*@'%.example.com' applies to *user_name* for any host in the example.com domain, and *user_name*@'192.168.1.%' applies to *user_name* for any host in the 192.168.1 class C subnet.

The simple form *user_name* is a synonym for *user_name*@'%'.

MySQL does not support wildcards in user names. To refer to an anonymous user, specify an account with an empty user name with the GRANT statement:

```
```

```
GRANT ALL ON test.* TO ''@'localhost' ...
```

In this case, any user who connects from the local host with the correct password for the anonymous user will be permitted access, with the privileges associated with the anonymous-user account.

For additional information about user name and host name values in account names, see Section 5.5.3, “Specifying Account Names”.

To specify quoted values, quote database, table, column, and routine names as identifiers. Quote user names and host names as identifiers or as strings. Quote passwords as strings. For string-quoting and identifier-quoting guidelines, see Section 8.1.1, “String Literals”, and Section 8.2, “Database, Table, Index, Column, and Alias Names”.

The “_” and “%” wildcards are permitted when specifying database names in `GRANT` statements that grant privileges at the global or database levels. This means, for example, that if you want to use a “_” character as part of a database name, you should specify it as “`\_`” in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern; for example, `GRANT ... ON `foo_bar`.* TO ...`.

Warning

If you permit anonymous users to connect to the MySQL server, you should also grant privileges to all local users as *`user_name`*`@localhost`. Otherwise, the anonymous user account for `localhost` in the `mysql.user` table (created during MySQL installation) is used when named users try to log in to the MySQL server from the local machine. For details, see Section 5.5.4, “Access Control, Stage 1: Connection Verification”.

To determine whether the preceding warning applies to you, execute the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

To avoid the problem just described, delete the local anonymous user account using this statement:

```
DROP USER ''@'localhost';
```

`GRANT` supports host names up to 60 characters long. Database, table, and column names can be up to 64 characters. User names can be up to 16 characters.

Warning

The permissible length for user names cannot be changed by altering the `mysql.user` table. Attempting to do so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server. You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure described in Section 4.4.5, “`mysql_fix_privilege_tables` — Upgrade MySQL System Tables”.

In MySQL 3.22.12 or later, if the account named in a `GRANT` statement does not exist in the `mysql.user` table, `GRANT` creates it. If you specify no `IDENTIFIED BY` clause or provide an empty password, the user has no password. *This is very insecure.*

When the `IDENTIFIED BY` clause is present and you have global grant privileges, the password becomes the new password for the account, even if the account exists and already has a password.

In the `IDENTIFIED BY` clause, the password should be given as the literal plaintext password value:

```
GRANT ... IDENTIFIED BY 'mypass';
```

To avoid specifying the plaintext password if you know its hash value (the value that `PASSWORD()` [868] would return for the password), specify the hash value preceded by the keyword `PASSWORD`:

```
GRANT ...
IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
```

For additional information about setting passwords, see Section 5.6.5, “Assigning Account Passwords”.

Important

`GRANT` may be recorded in server logs or in a history file such as `~/.mysql_history`, which means that plaintext passwords may be read by anyone having read access to that information. See Section 5.4.2, “Password Security in MySQL”.

Other Account Characteristics

The `WITH` clause is used for several purposes:

- To enable a user to grant privileges to other users
- To specify resource limits for a user
- To specify whether and how a user must use secure connections to the server

The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level. You should be careful to whom you give the `GRANT OPTION` [491] privilege because two users with different privileges may be able to combine privileges!

You cannot grant another user a privilege which you yourself do not have; the `GRANT OPTION` [491] privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the `GRANT OPTION` [491] privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the `INSERT` [492] privilege on a database. If you then grant the `SELECT` [492] privilege on the database and specify `WITH GRANT OPTION`, that user can give to other users not only the `SELECT` [492] privilege, but also `INSERT` [492]. If you then grant the `UPDATE` [493] privilege to the user on the database, the user can grant `INSERT` [492], `SELECT` [492], and `UPDATE` [493].

For a nonadministrative user, you should not grant the `ALTER` [491] privilege globally or for the `mysql` database. If you do that, the user can try to subvert the privilege system by renaming tables!

For additional information about security risks associated with particular privileges, see Section 5.5.1, “Privileges Provided by MySQL”.

Several `WITH` clause options specify limits on use of server resources by an account. The `MAX_QUERIES_PER_HOUR` *`count`*, `MAX_UPDATES_PER_HOUR` *`count`*, and `MAX_CONNECTIONS_PER_HOUR` *`count`* limits were implemented in MySQL 4.0.2. They restrict the number of queries, updates, and connections to the server permitted to this account during any given one-hour period. (Queries for which results are served from the query cache do not count against the `MAX_QUERIES_PER_HOUR` limit.) If *`count`* is `0` (the default), this means that there is no limitation for the account.

To specify resource limits for an existing user without affecting existing privileges, use `GRANT USAGE` at the global level (`ON *.*`) and name the limits to be changed. For example:

```
GRANT USAGE ON *.* TO ...
  WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

Limits not specified retain their current values.

For more information on restricting access to server resources, see Section 5.6.4, “Setting Account Resource Limits”.

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the user name and password. To specify SSL-related options for a MySQL account, use the `REQUIRE` clause of the `GRANT` statement. (For background information on the use of SSL with MySQL, see Section 5.6.6, “Using SSL for Secure Connections”.)

There are a number of different possibilities for limiting connection types for a given account:

- `REQUIRE NONE` indicates that the account has no SSL or X509 requirements. This is the default if no SSL-related `REQUIRE` options are specified. Unencrypted connections are permitted if the user name and password are valid. However, encrypted connections can also be used, at the client's option, if the client has the proper certificate and key files. That is, the client need not specify any SSL command options, in which case the connection will be unencrypted. To use an encrypted connection, the client must specify either the `--ssl-ca` [522] option, or all three of the `--ssl-ca` [522], `--ssl-key` [522], and `--ssl-cert` [522] options.

- The `REQUIRE SSL` option tells the server to permit only SSL-encrypted connections for the account.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    IDENTIFIED BY 'goodsecret' REQUIRE SSL;
  ```

  To connect, the client must specify the `--ssl-ca` [522] option, and may additionally specify the `--ssl-key` [522] and `--ssl-cert` [522] options.

- `REQUIRE X509` means that the client must have a valid certificate but that the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    IDENTIFIED BY 'goodsecret' REQUIRE X509;
  ```

  To connect, the client must specify the `--ssl-ca` [522], `--ssl-key` [522], and `--ssl-cert` [522] options. This is also true for `ISSUER` and `SUBJECT` because those `REQUIRE` options imply `X509`.

- `REQUIRE ISSUER 'issuer'` places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA `'issuer'`. If the client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    IDENTIFIED BY 'goodsecret'
    REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
      O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
  ```

  Note that the `'issuer'` value should be entered as a single string.

- `REQUIRE SUBJECT 'subject'` places the restriction on connection attempts that the client must present a valid X509 certificate containing the subject *`subject`*. If the client presents a certificate that is valid but has a different subject, the server rejects the connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
    O=MySQL demo client certificate/
    CN=Tonu Samuel/Email=tonu@example.com';
```

  Note that the `'subject'` value should be entered as a single string. MySQL does a simple string comparison of this value to the value in the certificate, so lettercase and component ordering must be given exactly as present in the certificate.

- `REQUIRE CIPHER 'cipher'` is needed to ensure that ciphers and key lengths of sufficient strength are used. SSL itself can be weak if old algorithms using short encryption keys are used. Using this option, you can ask that a specific cipher method is used for a connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The `SUBJECT`, `ISSUER`, and `CIPHER` options can be combined in the `REQUIRE` clause like this:

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
    O=MySQL demo client certificate/
    CN=Tonu Samuel/Email=tonu@example.com'
  AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
    O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The order of the options does not matter, but no option can be specified twice. Starting from MySQL 4.0.4, the `AND` keyword is optional between `REQUIRE` options.

If you are using table or column privileges for even one user, the server examines table and column privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

MySQL and Standard SQL Versions of `GRANT`

The biggest differences between the MySQL and standard SQL versions of `GRANT` are:

- MySQL associates privileges with the combination of a host name and user name and not with only a user name.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL does not support the standard SQL `UNDER` privilege, and does not support the `TRIGGER` privilege until MySQL 5.1.6.
- Standard SQL privileges are structured in a hierarchical manner, which means that if you remove a user, all privileges the user has been granted are revoked. This is not the case in MySQL 4.1 and earlier versions, where the granted privileges are not automatically revoked and you must revoke them explicitly. See Section 12.4.1.1, “`DROP USER` Syntax”.

- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped only with explicit `DROP USER` or `REVOKE` statements or by manipulating the MySQL grant tables directly.
- In MySQL, it is possible to have the `INSERT` [492] privilege for only some of the columns in a table. In this case, you can still execute `INSERT` statements on the table, provided that you insert values only for those columns for which you have the `INSERT` [492] privilege. The omitted columns are set to their implicit default values. (Standard SQL requires you to have the `INSERT` [492] privilege on all columns.) Section 10.1.4, “Data Type Default Values”, discusses implicit default values.

12.4.1.3 `REVOKE` Syntax

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON priv_level
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...
```

The `REVOKE` statement enables system administrators to revoke privileges from MySQL accounts. `REVOKE` is implemented in MySQL 3.22.11 or later. For earlier MySQL versions, it does nothing. Each account name uses the format described in Section 5.5.3, “Specifying Account Names”. For example:

```
REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
```

If you specify only the user name part of the account name, a host name part of `'%'` is used.

For details on the levels at which privileges exist, the permissible *`priv_type`* and *`priv_level`* values, and the syntax for specifying users and passwords, see Section 12.4.1.2, “`GRANT` Syntax”

To use the first `REVOKE` syntax, you must have the `GRANT OPTION` [491] privilege, and you must have the privileges that you are revoking.

To make it easy to revoke all privileges, MySQL 4.1.2 has added the following syntax, which drops all global, database, table, and column privileges for the named users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this `REVOKE` syntax, you must have the `UPDATE` [493] privilege for the `mysql` database.

Before MySQL 4.1.2, all privileges cannot be dropped at once. Two statements are necessary:

```
REVOKE ALL PRIVILEGES ON *.* FROM user [, user] ...
REVOKE GRANT OPTION ON *.* FROM user [, user] ...
```

`REVOKE` removes privileges, but does not drop `mysql.user` table entries. To remove a user account entirely, use `DELETE`. As of MySQL 4.1.1, you can also use `DROP USER` to remove users; see Section 12.4.1.1, “`DROP USER` Syntax”.

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` [418] system variable is set to a nonzero value, `REVOKE` cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create

such rows when `lower_case_table_names` [418] is set, but such rows might have been created prior to setting the variable.)

To verify an account's privileges after a `REVOKE` operation, use `SHOW GRANTS`. See Section 12.4.5.12, "`SHOW GRANTS` Syntax".

12.4.1.4 `SET PASSWORD` Syntax

```
SET PASSWORD [FOR user] =
    {
        PASSWORD('some password')
      | OLD_PASSWORD('some password')
      | 'encrypted password'
    }
```

The `SET PASSWORD` statement assigns a password to an existing MySQL user account.

If the password is specified using the `PASSWORD()` [868] or `OLD_PASSWORD()` [868] function, the literal text of the password should be given. If the password is specified without using either function, the password should be the already-encrypted password value as returned by `PASSWORD()` [868].

With no `FOR` clause, this statement sets the password for the current user. Any client that has connected to the server using a nonanonymous account can change the password for that account.

With a `FOR` clause, this statement sets the password for a specific account on the current server host. Only clients that have the `UPDATE` privilege for the `mysql` database can do this. The *`user`* value should be given in *`user_name@host_name`* format, where *`user_name`* and *`host_name`* are exactly as they are listed in the `User` and `Host` columns of the `mysql.user` table entry. For example, if you had an entry with `User` and `Host` column values of `'bob'` and `'%.loc.gov'`, you would write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

That is equivalent to the following statements:

```
UPDATE mysql.user SET Password=PASSWORD('newpass')
  WHERE User='bob' AND Host='%.loc.gov';
FLUSH PRIVILEGES;
```

Another way to set the password is to use `GRANT`:

```
GRANT USAGE ON *.* TO 'bob'@'%.loc.gov' IDENTIFIED BY 'newpass';
```

Important

`SET PASSWORD` may be recorded in server logs or in a history file such as `~/.mysql_history`, which means that plaintext passwords may be read by anyone having read access to that information. See Section 5.4.2, "Password Security in MySQL".

Note

If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not use the preceding `SET PASSWORD` or `UPDATE` statement without reading Section 5.4.2.3, "Password Hashing in MySQL", first. The password format changed in MySQL 4.1, and under certain circumstances it is possible that if you change your password, you might not be able to connect to the server afterward.

Starting from MySQL 4.1, to see which account the server authenticated you as, invoke the `CURRENT_USER()` [872] function.

For more information about setting passwords, see Section 5.6.5, “Assigning Account Passwords”

12.4.2 Table Maintenance Statements

12.4.2.1 `ANALYZE TABLE` Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

`ANALYZE TABLE` analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock for `MyISAM` and is locked with a read lock for `MyISAM`, `BDB`, and `InnoDB`. This statement works with `MyISAM`, `BDB`, and (as of MySQL 4.0.13) `InnoDB` tables. For `MyISAM` tables, this statement is equivalent to using `myisamchk --analyze`.

For more information on how the analysis works within `InnoDB`, see Section 13.2.15, “Restrictions on `InnoDB` Tables”.

MySQL uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

This statement requires `SELECT` [492] and `INSERT` [492] privileges for the table.

`ANALYZE TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `analyze`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

You can check the stored key distribution with the `SHOW INDEX` statement. See Section 12.4.5.13, “`SHOW INDEX` Syntax”.

If the table has not changed since the last `ANALYZE TABLE` statement, the table is not analyzed again.

Before MySQL 4.1.1, `ANALYZE TABLE` statements are not written to the binary log. As of MySQL 4.1.1, `ANALYZE TABLE` statements are written to the binary log so that they will be replicated to replication slaves. Logging can be suppressed with the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

12.4.2.2 `BACKUP TABLE` Syntax

```
BACKUP TABLE tbl_name [, tbl_name] ... TO '/path/to/backup/directory'
```

Note

This statement is deprecated and is removed in MySQL 5.5. As an alternative, `mysqldump` or `mysqlhotcopy` can be used instead.

`BACKUP TABLE` copies to the backup directory the minimum number of table files needed to restore the table, after flushing any buffered changes to disk. The statement works only for `MyISAM` tables. It

copies the `.frm` definition and `.MYD` data files. The `.MYI` index file can be rebuilt from those two files. The directory should be specified as a full path name. To restore the table, use `RESTORE TABLE`.

During the backup, a read lock is held for each table, one at time, as they are being backed up. If you want to back up several tables as a snapshot (preventing any of them from being changed during the backup operation), issue a `LOCK TABLES` statement first, to obtain a read lock for all tables in the group.

`BACKUP TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `backup`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

`BACKUP TABLE` is available in MySQL 3.23.25 and later.

12.4.2.3 `CHECK TABLE` Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

`CHECK TABLE` checks a table or tables for errors. `CHECK TABLE` works for `MyISAM` and `InnoDB` tables. For `MyISAM` tables, the key statistics are updated as well.

`CHECK TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `check`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

Note that the statement might produce many rows of information for each checked table. The last row has a `Msg_type` value of `status` and the `Msg_text` normally should be `OK`. If you don't get `OK`, or `Table is already up to date` you should normally run a repair of the table. See Section 6.6, “`MyISAM` Table Maintenance and Crash Recovery”. `Table is already up to date` means that the storage engine for the table indicated that there was no need to check the table.

The different check options that can be given are shown in the following table. These options are passed to the storage engine, which may use them or not. `MyISAM` uses them; they are ignored for `InnoDB` tables.

Type	Meaning
`QUICK`	Do not scan the rows to check for incorrect links.
`FAST`	Check only tables that have not been closed properly.
`CHANGED`	Check only tables that have been changed since the last check or that have not been closed properly.
`MEDIUM`	Scan rows to verify that deleted links are valid. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys.

Type	Meaning
EXTENDED	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time.

If none of the options QUICK, MEDIUM, or EXTENDED are specified, the default check type for dynamic-format MyISAM tables is MEDIUM. This has the same result as running myisamchk --medium-check *tbl_name* on the table. The default check type also is MEDIUM for static-format MyISAM tables, unless CHANGED or FAST is specified. In that case, the default is QUICK. The row scan is skipped for CHANGED and FAST because the rows are very seldom corrupted.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note

In some cases, CHECK TABLE changes the table. This happens if the table is marked as "corrupted" or "not closed properly" but CHECK TABLE does not find any problems in the table. In this case, CHECK TABLE marks the table as okay.

If a table is corrupted, it is most likely that the problem is in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

If you just want to check a table that you assume is okay, you should use no check options or the QUICK option. The latter should be used when you are in a hurry and can take the very small risk that QUICK does not find an error in the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as "corrupted" and cannot be used until it is repaired.)

FAST and CHANGED are mostly intended to be used from a script (for example, to be executed from cron) if you want to check tables from time to time. In most cases, FAST is to be preferred over CHANGED. (The only case when it is not preferred is when you suspect that you have found a bug in the MyISAM code.)

EXTENDED is to be used only after you have run a normal check but still get strange errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Use of CHECK TABLE ... EXTENDED might influence the execution plan generated by the query optimizer.

Some problems reported by CHECK TABLE cannot be corrected automatically:

- Found row where the auto_increment column has the value 0.

  This means that you have a row in the table where the AUTO_INCREMENT index column contains the value 0. (It is possible to create a row where the AUTO_INCREMENT column is 0 by explicitly setting the column to 0 with an UPDATE statement.)

  This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an ALTER TABLE on the table. In this case, the AUTO_INCREMENT column changes value according to the rules of AUTO_INCREMENT columns, which could cause problems such as a duplicate-key error.

  To get rid of the warning, simply execute an UPDATE statement to set the column to some value other than 0.

12.4.2.4 CHECKSUM TABLE Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

`CHECKSUM TABLE` reports a table checksum.

With `QUICK`, the live table checksum is reported if it is available, or `NULL` otherwise. This is very fast. A live checksum is enabled by specifying the `CHECKSUM=1` table option when you create the table; currently, this is supported only for `MyISAM` tables. See Section 12.1.5, "`CREATE TABLE` Syntax".

With `EXTENDED`, the entire table is read row by row and the checksum is calculated. This can be very slow for large tables.

If neither `QUICK` nor `EXTENDED` is specified, MySQL returns a live checksum if the table storage engine supports it and scans the table otherwise.

For a nonexistent table, `CHECKSUM TABLE` returns `NULL`.

The checksum value depends on the table row format. If the row format changes, the checksum also changes. For example, the storage format for `VARCHAR` changed between MySQL 4.1 and 5.0, so if a 4.1 table is upgraded to MySQL 5.0, the checksum value may change.

This statement is implemented in MySQL 4.1.1.

Important

If the checksums for two tables are different, then it is almost certain that the tables are different in some way. However, because the hashing function used by `CHECKSUM TABLE` is not guaranteed to be collision-free, there is a slight chance that two tables which are not identical can produce the same checksum.

12.4.2.5 `OPTIMIZE TABLE` Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

`OPTIMIZE TABLE` should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have `VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT` columns). Deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions. You can use `OPTIMIZE TABLE` to reclaim the unused space and to defragment the data file. After extensive changes to a table, this statement may also improve performance of statements that use the table, sometimes significantly.

This statement requires `SELECT` [492] and `INSERT` [492] privileges for the table.

`OPTIMIZE TABLE` works *only* for `MyISAM`, `BDB`, and `InnoDB` tables. It does *not* work for tables created using any other storage engine.

For `MyISAM` tables, `OPTIMIZE TABLE` works as follows:

1. If the table has deleted or split rows, repair the table.
2. If the index pages are not sorted, sort them.
3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

For `BDB` tables, `OPTIMIZE TABLE` currently is mapped to `ANALYZE TABLE`. See Section 12.4.2.1, "`ANALYZE TABLE` Syntax".

That was also the case for `InnoDB` tables before MySQL 4.1.3. As of 4.1.3, `OPTIMIZE TABLE` is mapped to `ALTER TABLE`, which rebuilds the table to update index statistics and free unused space in the clustered index.

You can make `OPTIMIZE TABLE` work on other storage engines by starting `mysqld` with the `--skip-new` or `--safe-mode` [391] option. In this case, `OPTIMIZE TABLE` is just mapped to `ALTER TABLE`.

`OPTIMIZE TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `optimize`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

Note that MySQL locks the table during the time `OPTIMIZE TABLE` is running.

Before MySQL 4.1.1, `OPTIMIZE TABLE` statements are not written to the binary log. As of MySQL 4.1.1, `OPTIMIZE TABLE` statements are written to the binary log so that they will be replicated to replication slaves. Logging can be suppressed with the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

12.4.2.6 `REPAIR TABLE` Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
    [QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` repairs a possibly corrupted table. By default, it has the same effect as `myisamchk --recover tbl_name`. `REPAIR TABLE` works for `MyISAM` and for `ARCHIVE` tables. See Section 13.1, “The `MyISAM` Storage Engine”, and Section 13.7, “The `ARCHIVE` Storage Engine”.

This statement requires `SELECT` [492] and `INSERT` [492] privileges for the table.

Normally, you should never have to run this statement. However, if disaster strikes, `REPAIR TABLE` is very likely to get back all your data from a `MyISAM` table. If tables become corrupted often, you should try to find the reason for it and so to eliminate the need to use `REPAIR TABLE`. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”, and Section 13.1.4, “`MyISAM` Table Problems”.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

Warning

If the server crashes during a `REPAIR TABLE` operation, it is essential after restarting it that you immediately execute another `REPAIR TABLE` statement for the table before performing any other operations on it. In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario that underscores the value of making a backup first.

`REPAIR TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `repair`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

The `REPAIR TABLE` statement might produce many rows of information for each repaired table. The last row has a `Msg_type` value of `status` and `Msg_test` normally should be `OK`. If you do not get `OK` for a `MyISAM` table, you should try repairing it with `myisamchk --safe-recover`. (`REPAIR TABLE` does not implement all the options of `myisamchk`.) With `myisamchk --safe-recover`, you can also use options that `REPAIR TABLE` does not support, such as `--max-record-length` [317].

If you use the `QUICK` option, `REPAIR TABLE` tries to repair only the index file, and not the data file. This type of repair is like that done by `myisamchk --recover --quick`.

If you use the `EXTENDED` option, MySQL creates the index row by row instead of creating one index at a time with sorting. (Before MySQL 4.1, this might be better than sorting on fixed-length keys if you have long `CHAR` keys that compress very well.) This type of repair is like that done by `myisamchk --safe-recover`.

As of MySQL 4.0.2, the `USE_FRM` option is available for use if the `.MYI` index file is missing or if its header is corrupted. This option tells MySQL not to trust the information in the `.MYI` file header and to re-create it using information from the `.frm` file. This kind of repair cannot be done with `myisamchk`.

Note

Use the `USE_FRM` option *only* if you cannot use regular `REPAIR` modes! Telling the server to ignore the `.MYI` file makes important table metadata stored in the `.MYI` unavailable to the repair process, which can have deleterious consequences:

- The current `AUTO_INCREMENT` value is lost.
- The link to deleted records in the table is lost, which means that free space for deleted records will remain unoccupied thereafter.
- The `.MYI` header indicates whether the table is compressed. If the server ignores this information, it cannot tell that a table is compressed and repair can cause change or loss of table contents. This means that `USE_FRM` should not be used with compressed tables. That should not be necessary, anyway: Compressed tables are read only, so they should not become corrupt.

Caution

Do not use `USE_FRM` if your table was created by a different version of the MySQL server than the one you are currently running. Doing so risks the loss of all rows in the table. It is particularly dangerous to use `USE_FRM` after the server returns this message:

```
Table upgrade required. Please do
"REPAIR TABLE `tbl_name`" to fix it!
```

Before MySQL 4.1.1, `REPAIR TABLE` statements are not written to the binary log. As of MySQL 4.1.1, `REPAIR TABLE` statements are written to the binary log so that they will be replicated to replication slaves. Logging can be suppressed with the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

You may be able to increase `REPAIR TABLE` performance by setting certain system variables. See Section 7.3.2.4, "Speed of `REPAIR TABLE` Statements".

12.4.2.7 `RESTORE TABLE` Syntax

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/directory'
```

Note

This statement is deprecated and is removed in MySQL 5.5.

`RESTORE TABLE` restores the table or tables from a backup that was made with `BACKUP TABLE`. The directory should be specified as a full path name.

Existing tables are not overwritten; if you try to restore over an existing table, an error occurs. Just as for `BACKUP TABLE`, `RESTORE TABLE` currently works only for `MyISAM` tables. Restored tables are not replicated from master to slave.

The backup for each table consists of its `.frm` format file and `.MYD` data file. The restore operation restores those files, and then uses them to rebuild the `.MYI` index file. Restoring takes longer than backing up due to the need to rebuild the indexes. The more indexes the table has, the longer it takes.

`RESTORE TABLE` returns a result set with the following columns.

Column	Value
`Table`	The table name
`Op`	Always `restore`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

12.4.3 User-Defined Function Statements

12.4.3.1 `CREATE FUNCTION` Syntax for User-Defined Functions

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL}
    SONAME shared_library_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` [817] or `CONCAT()` [795].

`function_name` is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value.

`shared_library_name` is the basename of the shared object file that contains the code that implements the function. As of MySQL 4.1.25, the file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` [423] system variable. If the value of `plugin_dir` [423] is empty, the behavior that is used before 4.1.25 applies: The file must be located in a directory that is searched by your system's dynamic linker. For more information, see Section 18.2.2.5, "Compiling and Installing User-Defined Functions".

To create a function, you must have the `INSERT` [492] privilege for the `mysql` database. This is necessary because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name. If you do not have this table,

you should run the mysql_fix_privilege_tables script to create it. See Section 4.4.5, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”.

An active function is one that has been loaded with CREATE FUNCTION and not removed with DROP FUNCTION. All active functions are reloaded each time the server starts, unless you start mysqld with the --skip-grant-tables [392] option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see Section 18.2.2, “Adding a New User-Defined Function”. For the UDF mechanism to work, functions must be written in C or C++ (or another language that can use C calling conventions), your operating system must support dynamic loading and you must have compiled mysqld dynamically (not statically).

AGGREGATE is a new option for MySQL 3.23. An AGGREGATE function works exactly like a native MySQL aggregate (summary) function such as SUM or COUNT() [882]. For AGGREGATE to work, your mysql.func table must contain a type column. If your mysql.func table does not have this column, you should run the mysql_fix_privilege_tables script to create it.

Note

To upgrade the shared library associated with a UDF, issue a DROP FUNCTION statement, upgrade the shared library, and then issue a CREATE FUNCTION statement. If you upgrade the shared library first and then use DROP FUNCTION, the server may crash.

12.4.3.2 DROP FUNCTION Syntax

```
DROP FUNCTION function_name
```

This statement drops the user-defined function (UDF) named *function_name*.

To drop a function, you must have the DELETE [491] privilege for the mysql database. This is because DROP FUNCTION removes a row from the mysql.func system table that records the function's name, type, and shared library name.

Note

To upgrade the shared library associated with a UDF, issue a DROP FUNCTION statement, upgrade the shared library, and then issue a CREATE FUNCTION statement. If you upgrade the shared library first and then use DROP FUNCTION, the server may crash.

12.4.4 SET Syntax

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
      user_var_name = expr
    | [GLOBAL | SESSION] system_var_name = expr
    | [@@global. | @@session. | @@]system_var_name = expr
```

The SET statement assigns values to different types of variables that affect the operation of the server or your client. Older versions of MySQL employed SET OPTION, but this syntax is deprecated in favor of SET without OPTION.

This section describes use of SET for assigning values to system variables or user variables. For general information about these types of variables, see Section 5.1.3, “Server System Variables”, and Section 8.4,

"User-Defined Variables". System variables also can be set at server startup, as described in Section 5.1.4, "Using System Variables".

Some variants of SET syntax are used in other contexts:

- SET CHARACTER SET and SET NAMES assign values to character set and collation variables associated with the connection to the server. SET ONESHOT is used for replication. These variants are described later in this section.
- SET PASSWORD assigns account passwords. See Section 12.4.1.4, "SET PASSWORD Syntax".
- SET TRANSACTION ISOLATION LEVEL sets the isolation level for transaction processing. See Section 12.3.6, "SET TRANSACTION Syntax".

The following discussion shows the different SET syntaxes that you can use to set variables. The examples use the = [789] assignment operator, but you can also use the := [788] assignment operator for this purpose.

A user variable is written as *@var_name* and can be set as follows:

```
SET @var_name = expr;
```

As of MySQL 4.0.3, many system variables are dynamic and can be changed while the server runs by using the SET statement. For a list, see Section 5.1.4.2, "Dynamic System Variables". To change a system variable with SET, refer to it as *var_name*, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by GLOBAL or @@global.. The SUPER [493] privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by SESSION, @@session., or @@. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- LOCAL and @@local. are synonyms for SESSION and @@session..
- If no modifier is present, SET changes the session variable.

A SET statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent GLOBAL or SESSION modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The *@@var_name* syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.)

The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the SET GLOBAL statement).

To prevent incorrect usage, MySQL produces an error if you use SET GLOBAL with a variable that can only be used with SET SESSION or if you do not specify GLOBAL (or @@global.) when setting a global variable.

To set a SESSION variable to the GLOBAL value or a GLOBAL value to the compiled-in MySQL default value, use the DEFAULT keyword. For example, the following two statements are identical in setting the session value of max_join_size [419] to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to DEFAULT. In such cases, use of DEFAULT results in an error.

You can refer to the values of specific global or sesson system variables in expressions by using one of the @@-modifiers. For example, you can retrieve values in a SELECT statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as *@@var_name* (that is, when you do not specify @@global. or @@session.), MySQL returns the session value if it exists and the global value otherwise. (This differs from SET *@@var_name = value*, which always refers to the session value.)

Note

Some variables displayed by SHOW VARIABLES may not be available using SELECT *@@var_name* syntax; an Unknown system variable occurs. As a workaround in such cases, you can use SHOW VARIABLES LIKE '*var_name*'.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with SET at runtime. On the other hand, with SET you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

To display system variables names and values, use the SHOW VARIABLES statement. (See Section 12.4.5.25, “SHOW VARIABLES Syntax”.)

The following list describes SET options that have nonstandard syntax (that is, options that are not set with *name = value* syntax).

- CHARACTER SET {*charset_name* | DEFAULT}

  This maps all strings from and to the client with the given mapping. Before MySQL 4.1, the only permissible value for *charset_name* is cp1251_koi8, but you can add

new mappings by editing the sql/convert.cc file in the MySQL source distribution.
As of MySQL 4.1.1, SET CHARACTER SET sets three session system variables: character_set_client [408] and character_set_results [408] are set to the given character set, and character_set_connection [408] to the value of character_set_database [408]. See Section 9.1.4, “Connection Character Sets and Collations”.

The default mapping can be restored by using the value DEFAULT. The default depends on the server configuration.

ucs2 cannot be used as a client character set, which means that it does not work for SET CHARACTER SET.

- NAMES {'*charset_name*' [COLLATE '*collation_name*'] | DEFAULT}

  SET NAMES sets the three session system variables character_set_client [408], character_set_connection [408], and character_set_results [408] to the given character set. Setting character_set_connection [408] to charset_name also sets collation_connection [409] to the default collation for charset_name. The optional COLLATE clause may be used to specify a collation explicitly. See Section 9.1.4, “Connection Character Sets and Collations”.

  The default mapping can be restored by using a value of DEFAULT. The default depends on the server configuration.

  ucs2 cannot be used as a client character set, which means that it does not work for SET NAMES.

  SET NAMES is available as of MySQL 4.1.0.

- ONE_SHOT

  This option is a modifier, not a variable. It is *only* for internal use for replication: mysqlbinlog uses SET ONE_SHOT to modify temporarily the values of character set, collation, and time zone variables to reflect at rollforward what they were originally. ONE_SHOT is available as of MySQL 4.1.3.

  ONE_SHOT is intended for use only with the permitted set of variables. With other variables, an error occurs:

```
mysql> SET ONE_SHOT max_allowed_packet = 1;
ERROR 1382 (HY000): The 'SET ONE_SHOT' syntax is reserved for purposes
internal to the MySQL server
```

  If ONE_SHOT is used with the permitted variables, it changes the variables as requested, but only for the next non-SET statement. After that, the server resets all character set, collation, and time zone-related system variables to their previous values. Example:

```
mysql> SET ONE_SHOT character_set_connection = latin5;

mysql> SET ONE_SHOT collation_connection = latin5_turkish_ci;

mysql> SHOW VARIABLES LIKE '%_connection';
+--------------------------+-------------------+
| Variable_name            | Value             |
+--------------------------+-------------------+
| character_set_connection | latin5            |
| collation_connection     | latin5_turkish_ci |
+--------------------------+-------------------+

mysql> SHOW VARIABLES LIKE '%_connection';
```

```
+--------------------------+-------------------+
| Variable_name            | Value             |
+--------------------------+-------------------+
| character_set_connection | latin1            |
| collation_connection     | latin1_swedish_ci |
+--------------------------+-------------------+
```

12.4.5 SHOW Syntax

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [LIKE 'pattern']
SHOW COLLATION [LIKE 'pattern']
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE TABLE tbl_name
SHOW DATABASES [LIKE 'pattern']
SHOW ENGINE engine_name {LOGS | STATUS }
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW [BDB] LOGS
SHOW MASTER STATUS
SHOW OPEN TABLES
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW TABLES [FROM db_name] [LIKE 'pattern']
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
SHOW WARNINGS [LIMIT [offset,] row_count]
```

If the syntax for a given SHOW statement includes a LIKE 'pattern' [804] part, 'pattern' is a string that can contain the SQL "%" and "_" wildcard characters. The pattern is useful for restricting statement output to matching values.

Many MySQL APIs (such as PHP) enable you to treat the result returned from a SHOW statement as you would a result set from a SELECT; see Chapter 17, *Connectors and APIs*, or your API documentation for more information.

12.4.5.1 SHOW BINARY LOGS Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in Section 12.5.1.1, "PURGE BINARY LOGS Syntax", that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
```

```
| binlog.000016 |    733481 |
+---------------+-----------+
```

SHOW MASTER LOGS was added in MySQL 3.23.38. As of MySQL 4.1.1, you can also use SHOW BINARY LOGS, which is equivalent. The File_size column is displayed as of MySQL 5.0.7.

12.4.5.2 SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify '*log_name*', the first binary log is displayed.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 12.2.7, "SELECT Syntax".

This statement is available as of MySQL 4.0.

Note

Issuing a SHOW BINLOG EVENTS with no LIMIT clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to SHOW BINLOG EVENTS, use the mysqlbinlog utility to save the binary log to a text file for later examination and analysis. See Section 4.6.6, "mysqlbinlog — Utility for Processing Binary Log Files".

Note

Some events relating to the setting of user and system variables are not included in the output from SHOW BINLOG EVENTS. To get complete coverage of events within a binary log, use mysqlbinlog.

12.4.5.3 SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET [LIKE 'pattern']
```

The SHOW CHARACTER SET statement shows all available character sets. It takes an optional LIKE [804] clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

The Maxlen column shows the maximum number of bytes required to store one character.

SHOW CHARACTER SET is available as of MySQL 4.1.0.

12.4.5.4 SHOW COLLATION Syntax

```
SHOW COLLATION [LIKE 'pattern']
```

This statement lists collations supported by the server. By default, the output from `SHOW COLLATION` includes all available collations. It takes an optional `LIKE` [804] clause whose *`pattern`* indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

The `Collation` and `Charset` columns indicate the names of the collation and the character set with which it is associated. `Id` is the collation ID. `Default` indicates whether the collation is the default for its character set. `Compiled` indicates whether the character set is compiled into the server. `Sortlen` is related to the amount of memory required to sort strings expressed in the character set.

To see the default collation for each character set, use the following statement. `Default` is a reserved word, so to use it as an identifier, it must be quoted as such:

```
mysql> SHOW COLLATION WHERE `Default` = 'Yes';
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| big5_chinese_ci     | big5     |  1 | Yes     | Yes      |       1 |
| dec8_swedish_ci     | dec8     |  3 | Yes     | Yes      |       1 |
| cp850_general_ci    | cp850    |  4 | Yes     | Yes      |       1 |
| hp8_english_ci      | hp8      |  6 | Yes     | Yes      |       1 |
| koi8r_general_ci    | koi8r    |  7 | Yes     | Yes      |       1 |
| latin1_swedish_ci   | latin1   |  8 | Yes     | Yes      |       1 |
...
```

`SHOW COLLATION` is available as of MySQL 4.1.0.

12.4.5.5 `SHOW COLUMNS` Syntax

```
SHOW [FULL] COLUMNS {FROM | IN} tbl_name [{FROM | IN} db_name] [LIKE 'pattern']
```

`SHOW COLUMNS` displays information about the columns in a given table.

```
mysql> SHOW COLUMNS FROM City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  |      | PRI | NULL    | auto_increment |
| Name       | char(35) |      |     |         |                |
| Country    | char(3)  |      | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  |      |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

If the data types differ from what you expect them to be based on a CREATE TABLE statement, note that MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in Section 12.1.5.2, “Silent Column Specification Changes”.

The FULL keyword can be used from MySQL 3.23.32 on. It causes the output to include the privileges you have for each column. As of MySQL 4.1, FULL also causes any per-column collation and comments to be displayed.

You can use *db_name.tbl_name* as an alternative to the *tbl_name* FROM *db_name* syntax. In other words, these two statements are equivalent:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

SHOW COLUMNS displays the following values for each table column:

Field indicates the column name.

Type indicates the column data type.

Collation indicates the collation for nonbinary string columns, or NULL for other columns. This value is displayed only if you use the FULL keyword.

The Null field indicates whether NULL values can be stored in the column, with YES displayed when NULL values are permitted.

The Key field indicates whether the column is indexed:

- If Key is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
- If Key is PRI, the column is a PRIMARY KEY or is one of the columns in a multiple-column PRIMARY KEY.
- If Key is UNI, the column is the first column of a unique-valued index that cannot contain NULL values.
- If Key is MUL, multiple occurrences of a given value are permitted within the column. The column is the first column of a nonunique index or a unique-valued index that can contain NULL values.

If more than one of the Key values applies to a given column of a table, Key displays the one with the highest priority, in the order PRI, UNI, MUL.

A UNIQUE index may be displayed as PRI if it cannot contain NULL values and there is no PRIMARY KEY in the table. A UNIQUE index may display as MUL if several columns form a composite UNIQUE index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

If the column permits NULL values, the Key value can be MUL even when a UNIQUE index is used. The rationale is that multiple rows in a UNIQUE index can hold a NULL value if the column is not declared NOT NULL. (This behavior changes in MySQL 5.0.)

The Default field indicates the default value that is assigned to the column.

The Extra field contains any additional information that is available about a given column. The value is auto_increment for columns that have the AUTO_INCREMENT attribute and empty otherwise.

Privileges indicates the privileges you have for the column. This value is displayed only if you use the FULL keyword.

Comment indicates any comment the column has. This value is displayed only if you use the FULL keyword.

SHOW FIELDS is a synonym for SHOW COLUMNS. You can also list a table's columns with the mysqlshow *db_name tbl_name* command.

The DESCRIBE statement provides information similar to SHOW COLUMNS. See Section 12.7.1, "DESCRIBE Syntax".

The SHOW CREATE TABLE, SHOW TABLE STATUS, and SHOW INDEX statements also provide information about tables. See Section 12.4.5, "SHOW Syntax".

12.4.5.6 SHOW CREATE DATABASE Syntax

```
SHOW CREATE DATABASE db_name
```

Shows the CREATE DATABASE statement that creates the given database. It was added in MySQL 4.1.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

SHOW CREATE DATABASE quotes table and column names according to the value of the sql_quote_show_create [430] option. See Section 5.1.3, "Server System Variables".

12.4.5.7 SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows the CREATE TABLE statement that creates the given table. The statement requires the SELECT [492] privilege for the table. It was added in MySQL 3.23.20.

```
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM
```

SHOW CREATE TABLE quotes table and column names according to the value of the sql_quote_show_create [430] option. See Section 5.1.3, "Server System Variables".

12.4.5.8 SHOW DATABASES Syntax

```
SHOW DATABASES [LIKE 'pattern']
```

SHOW DATABASES lists the databases on the MySQL server host. As of MySQL 4.0.2, you see only those databases for which you have some kind of privilege, if you do not have the global SHOW DATABASES privilege. You can also get this list using the mysqlshow command.

If the server was started with the --skip-show-database [394] option, you cannot use this statement at all unless you have the SHOW DATABASES [492] privilege.

MySQL implements databases as directories in the data directory, so this statement simply lists directories in that location. However, the output may include names of directories that do not correspond to actual databases.

12.4.5.9 SHOW ENGINE Syntax

```
SHOW ENGINE engine_name {LOGS | STATUS }
```

SHOW ENGINE displays log or status information about a storage engine. The following statements currently are supported:

```
SHOW ENGINE BDB LOGS
SHOW ENGINE INNODB STATUS
SHOW ENGINE NDB STATUS
SHOW ENGINE NDBCLUSTER STATUS
```

SHOW ENGINE BDB LOGS displays status information about existing BDB log files. It returns the following fields:

- File

  The full path to the log file.

- Type

  The log file type (BDB for Berkeley DB log files).

- Status

  The status of the log file (FREE if the file can be removed, or IN USE if the file is needed by the transaction subsystem)

SHOW ENGINE INNODB STATUS displays extensive information from the standard InnoDB Monitor about the state of the InnoDB storage engine. For information about the standard monitor and other InnoDB Monitors that provide information about InnoDB processing, see Section 13.2.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”.

Older (and now deprecated) synonyms for these statements are SHOW [BDB] LOGS and SHOW INNODB STATUS.

SHOW ENGINE can be used as of MySQL 4.1.2.

Beginning with MySQL 4.1.3, if the server has the NDBCLUSTER storage engine enabled, SHOW ENGINE NDB STATUS can be used to display cluster status information. Sample output from this statement is shown here:

```
mysql> SHOW ENGINE NDB STATUS;
+----------------------+---------+------+--------+
| free_list            | created | free | sizeof |
+----------------------+---------+------+--------+
| NdbTransaction       |       5 |    0 |    208 |
| NdbOperation         |       4 |    4 |    660 |
| NdbIndexScanOperation |       1 |    1 |    736 |
| NdbIndexOperation    |       0 |    0 |   1060 |
| NdbRecAttr           |     645 |  645 |     72 |
| NdbApiSignal         |      16 |   16 |    136 |
| NdbLabel             |       0 |    0 |    196 |
```

```
| NdbBranch             |       0 |    0 |     24 |
| NdbSubroutine         |       0 |    0 |     68 |
| NdbCall               |       0 |    0 |     16 |
| NdbBlob               |       2 |    2 |    204 |
| NdbReceiver           |       2 |    0 |     68 |
+-----------------------+---------+------+--------+
12 rows in set (0.00 sec)
```

The most useful of the rows from the output of this statement are described in the following list:

- `NdbTransaction`: The number and size of `NdbTransaction` objects that have been created. An `NdbTransaction` is created each time a table schema operation (such as `CREATE TABLE` or `ALTER TABLE`) is performed on an `NDB` table.
- `NdbOperation`: The number and size of `NdbOperation` objects that have been created.
- `NdbIndexScanOperation`: The number and size of `NdbIndexScanOperation` objects that have been created.
- `NdbIndexOperation`: The number and size of `NdbIndexOperation` objects that have been created.
- `NdbRecAttr`: The number and size of `NdbRecAttr` objects that have been created. In general, one of these is created each time a data manipulation statement is performed by an SQL node.
- `NdbBlob`: The number and size of `NdbBlob` objects that have been created. An `NdbBlob` is created for each new operation involving a `BLOB` column in an `NDB` table.
- `NdbReceiver`: The number and size of any `NdbReceiver` object that have been created. The number in the `created` column is the same as the number of data nodes in the cluster to which the MySQL server has connected.

Note

`SHOW ENGINE NDB STATUS` returns an empty result if no operations involving `NDB` tables have been performed by the MySQL client accessing the SQL node on which this statement is run.

`SHOW ENGINE NDBCLUSTER STATUS` is a synonym for `SHOW ENGINE NDB STATUS`.

12.4.5.10 SHOW ENGINES Syntax

```
SHOW [STORAGE] ENGINES
```

`SHOW ENGINES` displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. This statement is implemented in MySQL 4.1.2. `SHOW TABLE TYPES` is a synonym, but is deprecated and is removed in MySQL 5.5.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
*************************** 2. row ***************************
Engine: HEAP
Support: YES
Comment: Alias for MEMORY
*************************** 3. row ***************************
```

```
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 4. row ***************************
Engine: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
*************************** 5. row ***************************
Engine: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
*************************** 6. row ***************************
Engine: ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MyISAM
*************************** 7. row ***************************
Engine: MRG_ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MERGE
*************************** 8. row ***************************
Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
*************************** 9. row ***************************
Engine: INNOBASE
Support: YES
Comment: Alias for INNODB
*************************** 10. row ***************************
Engine: BDB
Support: YES
Comment: Supports transactions and page-level locking
*************************** 11. row ***************************
Engine: BERKELEYDB
Support: YES
Comment: Alias for BDB
*************************** 12. row ***************************
Engine: NDBCLUSTER
Support: NO
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
Engine: NDB
Support: NO
Comment: Alias for NDBCLUSTER
*************************** 14. row ***************************
Engine: EXAMPLE
Support: NO
Comment: Example storage engine
*************************** 15. row ***************************
Engine: ARCHIVE
Support: NO
Comment: Archive storage engine
*************************** 16. row ***************************
Engine: CSV
Support: NO
Comment: CSV storage engine
*************************** 17. row ***************************
Engine: BLACKHOLE
Support: NO
Comment: Storage engine designed to act as null storage
```

The `Support` value indicates whether the particular storage engine is supported, and which is the default engine. For example, if the server is started with the `--default-table-type=InnoDB` [386] option, the `Support` value for the `InnoDB` row has the value `DEFAULT`. See Chapter 13, *Storage Engines*.

All MySQL servers beginning with the 3.23 release series support `MyISAM` tables, because `MyISAM` is the default storage engine.

12.4.5.11 SHOW ERRORS Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

This statement is similar to SHOW WARNINGS, except that instead of displaying errors, warnings, and notes, it displays only errors. SHOW ERRORS is available as of MySQL 4.1.0.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 12.2.7, “SELECT Syntax”.

The SHOW COUNT(*) ERRORS statement displays the number of errors. You can also retrieve this number from the error_count [410] variable:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

For more information, see Section 12.4.5.26, “SHOW WARNINGS Syntax”.

12.4.5.12 SHOW GRANTS Syntax

```
SHOW GRANTS [FOR user]
```

This statement lists the GRANT statement or statements that must be issued to duplicate the privileges that are granted to a MySQL user account. The account is named using the same format as for the GRANT statement; for example, 'jeffrey'@'localhost'. If you specify only the user name part of the account name, a host name part of '%' is used. For additional information about specifying account names, see Section 12.4.1.2, “GRANT Syntax”.

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
```

As of MySQL 4.1.2, to list the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

Before MySQL 4.1.2, you can find out what user the session was authenticated as by selecting the value of the CURRENT_USER() [872] function (new in MySQL 4.0.6). Then use that value in the SHOW GRANTS statement. See Section 11.13, “Information Functions”.

SHOW GRANTS displays only the privileges granted explicitly to the named account. Other privileges might be available to the account, but they are not displayed. For example, if an anonymous account exists, the named account might be able to use its privileges, but SHOW GRANTS will not display them.

SHOW GRANTS requires the SELECT [492] privilege for the mysql database.

SHOW GRANTS is available as of MySQL 3.23.4.

12.4.5.13 SHOW INDEX Syntax

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
```

SHOW INDEX returns table index information. The format resembles that of the SQLStatistics call in ODBC.

SHOW INDEX returns the following fields:

- Table

  The name of the table.

- Non_unique

  0 if the index cannot contain duplicates, 1 if it can.

- Key_name

  The name of the index.

- Seq_in_index

  The column sequence number in the index, starting with 1.

- Column_name

  The column name.

- Collation

  How the column is sorted in the index. In MySQL, this can have values “A” (Ascending) or NULL (Not sorted).

- Cardinality

  An estimate of the number of unique values in the index. This is updated by running ANALYZE TABLE or myisamchk -a. Cardinality is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- Sub_part

  The number of indexed characters if the column is only partly indexed, NULL if the entire column is indexed.

- Packed

  Indicates how the key is packed. NULL if it is not.

- Null

  Contains YES if the column may contain NULL. If not, in MySQL 4.1 and earlier, the column contains an empty string ('').

- Index_type

The index method used (BTREE, FULLTEXT, HASH, RTREE).

- Comment

  Various remarks. Before MySQL 4.0.2 when the Index_type column was added, Comment indicates whether an index is FULLTEXT.

The Packed and Comment columns were added in MySQL 3.23.0. The Null and Index_type columns were added in MySQL 4.0.2.

You can use *db_name*.*tbl_name* as an alternative to the *tbl_name* FROM *db_name* syntax. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

You can also list a table's indexes with the mysqlshow -k *db_name* *tbl_name* command.

12.4.5.14 SHOW INNODB STATUS Syntax

```
SHOW INNODB STATUS
```

This statement shows extensive information about the state of the InnoDB storage engine. As of MySQL 4.1.2, it is deprecated and SHOW ENGINE INNODB STATUS should be used instead. See Section 12.4.5.9, “SHOW ENGINE Syntax”. SHOW INNODB STATUS is removed in MySQL 5.5.

12.4.5.15 SHOW LOGS Syntax

```
SHOW [BDB] LOGS
```

SHOW LOGS displays status information about existing BDB log files. It was implemented in MySQL 3.23.29. An alias for it (available as of MySQL 4.1.1) is SHOW BDB LOGS. As of MySQL 4.1.2, this statement is deprecated and SHOW ENGINE BDB LOGS should be used instead. See Section 12.4.5.9, “SHOW ENGINE Syntax”.

12.4.5.16 SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

This statement provides status information about the binary log files of the master. It requires either the SUPER [493] or REPLICATION CLIENT [492] privilege.

Example:

```
mysql> SHOW MASTER STATUS;
+--------------+----------+--------------+------------------+
| File         | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
```

12.4.5.17 SHOW OPEN TABLES Syntax

```
SHOW OPEN TABLES
```

`SHOW OPEN TABLES` lists the non-`TEMPORARY` tables that are currently open in the table cache. See Section 7.7.2, “How MySQL Opens and Closes Tables”.

`SHOW OPEN TABLES` returns the following columns:

- `Database`

  The database containing the table.

- `Table`

  The table name.

- `In_use`

  The number of table locks or lock requests there are for the table. For example, if one client acquires a lock for a table using `LOCK TABLE t1 WRITE`, `In_use` will be 1. If another client issues `LOCK TABLE t1 WRITE` while the table remains locked, the client will block waiting for the lock, but the lock request causes `In_use` to be 2. If the count is zero, the table is open but not currently being used. `In_use` is also increased by the `HANDLER ... OPEN` statement and decreased by `HANDLER ... CLOSE`.

- `Name_locked`

  Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

Before MySQL 4.0, `SHOW OPEN TABLES` displays information only for open tables in the default database, and the output format is somewhat different. The `Open_tables_in_`*`db_name`* column indicates the table name, and the `Comment` column displays all other available information.

`SHOW OPEN TABLES` was added in MySQL 3.23.33.

Note

It is not possible to guarantee the order in which the tables are displayed in this output of this statement from one invokation to the next.

12.4.5.18 `SHOW PRIVILEGES` Syntax

```
SHOW PRIVILEGES
```

`SHOW PRIVILEGES` shows the list of system privileges that the MySQL server supports. This statement is implemented as of MySQL 4.1.0. The exact list of privileges depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Alter
Context: Tables
Comment: To alter the table
*************************** 2. row ***************************
Privilege: Create temporary tables
Context: Databases
Comment: To use CREATE TEMPORARY TABLE
*************************** 3. row ***************************
Privilege: Create
Context: Databases,Tables,Indexes
```

```
Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Delete
Context: Tables
Comment: To delete existing rows
*************************** 5. row ***************************
Privilege: Drop
Context: Databases,Tables
Comment: To drop databases and tables
...
```

Privileges belonging to a specific user are displayed by the SHOW GRANTS statement. See Section 12.4.5.12, “SHOW GRANTS Syntax”, for more information.

12.4.5.19 SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

SHOW PROCESSLIST shows you which threads are running. You can also get this information using the mysqladmin processlist command. If you have the PROCESS [492] privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). If you do not use the FULL keyword, only the first 100 characters of each statement are shown in the Info field.

This statement is very useful if you get the “too many connections” error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the SUPER [493] privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the KILL statement. See Section 12.4.6.3, “KILL Syntax”.

Here is an example of what SHOW PROCESSLIST output looks like:

```
mysql> SHOW FULL PROCESSLIST\G
*************************** 1. row ***************************
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
*************************** 2. row ***************************
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave
       I/O thread to update it
Info: NULL
*************************** 3. row ***************************
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
```

```
*************************** 4. row ***************************
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
*************************** 5. row ***************************
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)
```

The columns have the following meaning:

- `Id`

  The connection identifier.

- `User`

  The MySQL user who issued the statement. If this is `system user`, it refers to a nonclient thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. For `system user`, there is no host specified in the `Host` column.

- `Host`

  The host name of the client issuing the statement (except for `system user` where there is no host). As of MySQL 4.0.12, `SHOW PROCESSLIST` reports the host name for TCP/IP connections in *`host_name:client_port`* format to make it easier to determine which client is doing what.

- `db`

  The default database, if one is selected, otherwise `NULL`.

- `Command`

  The type of command the thread is executing. For descriptions for thread commands, see Section 7.11, “Examining Thread Information”. The value of this column corresponds to the `COM_`*`xxx`* commands of the client/server protocol and `Com_`*`xxx`* status variables. See Section 5.1.5, “Server Status Variables”

- `Time`

  The time in seconds that the thread has been in its current state.

- `State`

  An action, event, or state that indicates what the thread is doing. Descriptions for `State` values can be found at Section 7.11, “Examining Thread Information”.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

For the SHOW PROCESSLIST statement, the value of State is NULL.

- Info

  The statement that the thread is executing, or NULL if it is not executing any statement.

12.4.5.20 SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of replication slaves currently registered with the master. Only slaves started with the --report-host=*host_name* [1178] option are visible in this list.

The list is displayed on any server (not just the master server). The output looks like this:

```
mysql> SHOW SLAVE HOSTS;
+------------+-----------+------+-----------+
| Server_id  | Host      | Port | Master_id |
+------------+-----------+------+-----------+
|  192168010 | iconnect2 | 3306 | 192168011 |
| 1921680101 | athena    | 3306 | 192168011 |
+------------+-----------+------+-----------+
```

- Server_id: The unique server ID of the slave server, as configured in the server's option file, or on the command line with --server-id=*value* [1167].
- Host: The host name of the slave server, as configured in the server's option file, or on the command line with --report-host=*host_name* [1178]. Note that this can differ from the machine name as configured in the operating system.
- Port: The port the slave server is listening on.
- Master_id: The unique server ID of the master server that the slave server is replicating from.

Some MySQL versions report another variable, Rpl_recovery_rank. This variable was never used, and was eventually removed.

12.4.5.21 SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS
```

This statement provides status information on essential parameters of the slave threads. It requires either the SUPER [493] or REPLICATION CLIENT [492] privilege.

If you issue this statement using the mysql client, you can use a \G statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
       Slave_IO_State: Waiting for master to send event
          Master_Host: localhost
          Master_User: root
          Master_Port: 3306
        Connect_Retry: 3
      Master_Log_File: gbichot-bin.005
  Read_Master_Log_Pos: 79
```

```
       Relay_Log_File: gbichot-relay-bin.005
        Relay_Log_Pos: 548
Relay_Master_Log_File: gbichot-bin.005
     Slave_IO_Running: Yes
    Slave_SQL_Running: Yes
      Replicate_Do_DB:
  Replicate_Ignore_DB:
           Last_Errno: 0
           Last_Error:
         Skip_Counter: 0
  Exec_Master_Log_Pos: 79
      Relay_Log_Space: 552
      Until_Condition: None
       Until_Log_File:
        Until_Log_Pos: 0
   Master_SSL_Allowed: No
   Master_SSL_CA_File:
   Master_SSL_CA_Path:
      Master_SSL_Cert:
    Master_SSL_Cipher:
       Master_SSL_Key:
Seconds_Behind_Master: 8
```

Depending on your version of MySQL, you may not see all the fields just shown. In particular, several fields are present only as of MySQL 4.1.1.

`SHOW SLAVE STATUS` returns the following fields:

- `Slave_IO_State`

  A copy of the `State` field of the `SHOW PROCESSLIST` output for the slave I/O thread. This tells you what the thread is doing: trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. Possible states are listed in Section 14.3, “Replication Implementation Details”. For versions of MySQL prior to 4.1.14, it is necessary to check this field for connection problems. In those versions, the thread could be running while unsuccessfully trying to connect to the master; only this field makes you aware of the problem. The state of the SQL thread is not copied because it is simpler. If it is running, there is no problem; if it is not, you can find the error in the `Last_Error` field (described later).

  This field is present beginning with MySQL 4.1.1.

- `Master_Host`

  The master host that the slave is connected to.

- `Master_User`

  The user name of the account used to connect to the master.

- `Master_Port`

  The port used to connect to the master.

- `Connect_Retry`

  The number of seconds between connect retries (default 60). This can be set with the `CHANGE MASTER TO` statement or `--master-connect-retry` [1173] option.

- `Master_Log_File`

  The name of the master binary log file from which the I/O thread is currently reading.

- Read_Master_Log_Pos

  The position in the current master binary log file up to which the I/O thread has read.

- Relay_Log_File

  The name of the relay log file from which the SQL thread is currently reading and executing.

- Relay_Log_Pos

  The position in the current relay log file up to which the SQL thread has read and executed.

- Relay_Master_Log_File

  The name of the master binary log file containing the most recent event executed by the SQL thread.

- Slave_IO_Running

  Whether the I/O thread is started and has connected successfully to the master. Internally, the state of this thread is represented by one of the following three values:

  - **MYSQL_SLAVE_NOT_RUN.** The slave I/O thread is not running. For this state, Slave_IO_Running is No.
  - **MYSQL_SLAVE_RUN_NOT_CONNECT.** The slave I/O thread is running, but is not connected to a replication master. For this state, Slave_IO_Running depends on the server version as shown in the following table.

MySQL Version	Slave_IO_Running
4.1 (4.1.13 and earlier); 5.0 (5.0.11 and earlier)	Yes
4.1 (4.1.14 and later); 5.0 (5.0.12 and later)	No
5.1, 5.4	No
5.5	Connecting

  - **MYSQL_SLAVE_RUN_CONNECT.** The slave I/O thread is running, and is connected to a replication master. For this state, Slave_IO_Running is Yes.

- Slave_SQL_Running

  Whether the SQL thread is started.

- Replicate_Do_DB, Replicate_Ignore_DB

  The lists of databases that were specified with the --replicate-do-db [1176] and --replicate-ignore-db [1176] options, if any.

  These fields are present beginning with MySQL 4.1.1.

- Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table

  The lists of tables that were specified with the --replicate-do-table [1177], --replicate-ignore-table [1177], --replicate-wild-do-table [1178], and --replicate-wild-ignore-table [1178] options, if any.

  These fields are present beginning with MySQL 4.1.1.

- `Last_Errno`, `Last_Error`

  The error number and error message returned by the most recently executed statement. An error number of 0 and message of the empty string mean “no error.” If the `Last_Error` value is not empty, it also appears as a message in the slave's error log. For example:

```
Last_Errno: 1051
Last_Error: error 'Unknown table 'z'' on query 'drop table z'
```

  The message indicates that the table `z` existed on the master and was dropped there, but it did not exist on the slave, so `DROP TABLE` failed on the slave. (This might occur, for example, if you forget to copy the table to the slave when setting up replication.)

  Note

  When the slave SQL thread receives an error, it reports the error first, then stops the SQL thread. This means that there is a small window of time during which `SHOW SLAVE STATUS` shows a nonzero value for `Last_Errno` even though `Slave_SQL_Running` still displays `Yes`.

- `Skip_Counter`

  The current value of the `sql_slave_skip_counter` [1181] system variable. See Section 12.5.2.6, “`SET GLOBAL SQL_SLAVE_SKIP_COUNTER` Syntax”.

- `Exec_Master_Log_Pos`

  The position in the current master binary file up to which the SQL thread has read and executed. The coordinates given by (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) in the master's binary log correspond to the coordinates given by (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

- `Relay_Log_Space`

  The total combined size of all existing relay log files.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`

  The values specified in the `UNTIL` clause of the `START SLAVE` statement.

  `Until_Condition` has these values:

  - `None` if no `UNTIL` clause was specified
  - `Master` if the slave is reading until a given position in the master's binary log
  - `Relay` if the slave is reading until a given position in its relay log

  `Until_Log_File` and `Until_Log_Pos` indicate the log file name and position that define the coordinates at which the SQL thread stops executing.

  These fields are present beginning with MySQL 4.1.1.

- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_Key`

  These fields show the SSL parameters used by the slave to connect to the master, if any.

  `Master_SSL_Allowed` has these values:

- `Yes` if an SSL connection to the master is permitted
- `No` if an SSL connection to the master is not permitted
- `Ignored` if an SSL connection is permitted but the slave server does not have SSL support enabled

The values of the other SSL-related fields correspond to the values of the `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_CIPHER`, and `MASTER_SSL_KEY` options to the `CHANGE MASTER TO` statement. See Section 12.5.2.1, “`CHANGE MASTER TO` Syntax”.

These fields are present beginning with MySQL 4.1.1.

- `Seconds_Behind_Master`

  This field is present beginning with MySQL 4.1.1. It is been experimental and has been changed in MySQL 4.1.9. The following applies to slaves running MySQL 4.1.9 or newer. This field is an indication of how “late” the slave is:

  - When the slave SQL thread is actively processing updates, this field is the number of seconds that have elapsed since the timestamp of the most recent event on the master executed by that thread.
  - When the SQL thread has caught up to the slave I/O thread and is idle waiting for more events from the I/O thread, this field is zero.

  In essence, this field measures the time difference in seconds between the slave SQL thread and the slave I/O thread.

  If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is *not* a good approximation; the slave SQL thread may quite often be caught up with the slow-reading slave I/O thread, so `Seconds_Behind_Master` often shows a value of 0, even if the I/O thread is late compared to the master. In other words, *this column is useful only for fast networks*.

  This time difference computation works even though the master and slave do not have identical clocks (the clock difference is computed when the slave I/O thread starts, and assumed to remain constant from then on). `Seconds_Behind_Master` is `NULL` (“unknown”) if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. For example, if the slave I/O thread is running but is not connected to the master and is sleeping for the number of seconds given by the `CHANGE MASTER TO` statement or `--master-connect-retry` [1173] option (default 60) before reconnecting, the value is `NULL`. This is because the slave cannot know what the master is doing, and so cannot say reliably how late it is.

  The value of this field is based on the timestamps stored in events, which are preserved through replication. This means that if a master M1 is itself a slave of M0, any event from M1's binary log that originates from M0's binary log has M0's timestamp for that event. This enables MySQL to replicate `TIMESTAMP` successfully. However, the problem for `Seconds_Behind_Master` is that if M1 also receives direct updates from clients, the `Seconds_Behind_Master` value randomly fluctuates because sometimes the last event from M1 originates from M0 and sometimes is the result of a direct update on M1.

12.4.5.22 `SHOW STATUS` Syntax

```
SHOW STATUS [LIKE 'pattern']
```

SHOW STATUS provides server status information. This information also can be obtained using the mysqladmin extended-status command. This statement does not require any privilege. It requires only the ability to connect to the server.

Partial output is shown here. The list of names and values may be different for your server. The meaning of each variable is given in Section 5.1.5, “Server Status Variables”.

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
| Bytes_sent               | 1176560426 |
| Connections              | 30023      |
| Created_tmp_disk_tables  | 0          |
| Created_tmp_tables       | 8340       |
| Created_tmp_files        | 60         |
...
| Open_tables              | 1          |
| Open_files               | 2          |
| Open_streams             | 0          |
| Opened_tables            | 44600      |
| Questions                | 2026873    |
...
| Table_locks_immediate    | 1920382    |
| Table_locks_waited       | 0          |
| Threads_cached           | 0          |
| Threads_created          | 30022      |
| Threads_connected        | 1          |
| Threads_running          | 1          |
| Uptime                   | 80380      |
+--------------------------+------------+
```

With a LIKE [804] clause, the statement displays only rows for those variables with names that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

12.4.5.23 SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS [{FROM | IN} db_name] [LIKE 'pattern']
```

SHOW TABLE STATUS works likes SHOW TABLES, but provides a lot of information about each non-TEMPORARY table. You can also get this list using the mysqlshow --status db_name command. This statement was added in MySQL 3.23.

SHOW TABLE STATUS returns the following fields:

- Name

The name of the table.

- Engine

  The storage engine for the table. See Chapter 13, *Storage Engines*. Before MySQL 4.1.2, this value is labeled as Type.

- Version

  The version number of the table's .frm file.

- Row_format

  The row-storage format (Fixed, Dynamic, Compressed, Redundant, Compact). For MyISAM tables, (Dynamic corresponds to what myisamchk -dvv reports as Packed. InnoDB tables are always in the Redundant format.

- Rows

  The number of rows. Some storage engines, such as MyISAM and ISAM, store the exact count. For other storage engines, such as InnoDB, this value is an approximation, and may vary from the actual value by as much as 40 to 50%. In such cases, use SELECT COUNT(*) to obtain an accurate count.

- Avg_row_length

  The average row length.

- Data_length

  The length of the data file.

- Max_data_length

  The maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

- Index_length

  The length of the index file.

- Data_free

  The number of allocated but unused bytes.

- Auto_increment

  The next AUTO_INCREMENT value.

- Create_time

  When the table was created.

- Update_time

  When the data file was last updated. For some storage engines, this value is NULL. For example, InnoDB stores multiple tables in its tablespace and the data file timestamp does not apply. For MyISAM, the data file timestamp is used; however, on Windows the timestamp is not updated by updates so the value is inaccurate.

- `Check_time`

  When the table was last checked. Not all storage engines update this time, in which case the value is always `NULL`.

- `Collation`

  The table's character set and collation. (Implemented in 4.1.1)

- `Checksum`

  The live checksum value (if any). (Implemented in 4.1.1)

- `Create_options`

  Extra options used with `CREATE TABLE`. The original options supplied when `CREATE TABLE` is called are retained and the options reported here may differ from the active table settings and options.

- `Comment`

  The comment used when creating the table (or information as to why MySQL could not access the table information).

In the table comment, `InnoDB` tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of completely free 1MB extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

For `MEMORY` (`HEAP`) tables, the `Data_length`, `Max_data_length`, and `Index_length` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

For views, all the fields displayed by `SHOW TABLE STATUS` are `NULL` except that `Name` indicates the view name and `Comment` says `view`.

12.4.5.24 SHOW TABLES Syntax

```
SHOW TABLES [{FROM | IN} db_name] [LIKE 'pattern']
```

`SHOW TABLES` lists the non-`TEMPORARY` tables in a given database. You can also get this list using the `mysqlshow db_name` command.

The output from `SHOW TABLES` contains a single column of table names.

If you have no privileges for a table, the table does not show up in the output from `SHOW TABLES` or `mysqlshow db_name`.

12.4.5.25 SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

`SHOW VARIABLES` shows the values of MySQL system variables. This information also can be obtained using the `mysqladmin variables` command. This statement does not require any privilege. It requires only the ability to connect to the server.

The GLOBAL and SESSION modifiers are new in MySQL 4.0.3. With the GLOBAL modifier, SHOW VARIABLES displays the values that are used for new connections to MySQL. With SESSION, it displays the values that are in effect for the current connection. If no modifier is present, the default is SESSION. LOCAL is a synonym for SESSION.

If the default system variable values are unsuitable, you can set them using command options when mysqld starts, and most can be changed at runtime with the SET statement. See Section 5.1.4, “Using System Variables”, and Section 12.4.4, “SET Syntax”.

Partial output is shown here. The list of names and values may be different for your server. Section 5.1.3, “Server System Variables”, describes the meaning of each variable, and Section 7.8.2, “Tuning Server Parameters”, provides information about tuning them.

```
mysql> SHOW VARIABLES;
+---------------------------------+-----------------------------+
| Variable_name                   | Value                       |
+---------------------------------+-----------------------------|
| back_log                        | 50                          |
| basedir                         | /usr/local/mysql            |
| bdb_cache_size                  | 8388572                     |
| bdb_log_buffer_size             | 32768                       |
| bdb_home                        | /usr/local/mysql            |
...
| max_connections                 | 100                         |
| max_connect_errors              | 10                          |
| max_delayed_threads             | 20                          |
| max_error_count                 | 64                          |
| max_heap_table_size             | 16777216                    |
| max_join_size                   | 4294967295                  |
| max_relay_log_size              | 0                           |
| max_sort_length                 | 1024                        |
...
| timezone                        | EEST                        |
| tmp_table_size                  | 33554432                    |
| tmpdir                          | /tmp/:/mnt/hd2/tmp/         |
| version                         | 4.1.18                      |
| wait_timeout                    | 28800                       |
+---------------------------------+-----------------------------+
```

With a LIKE [804] clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a LIKE [804] clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the “%” wildcard character in a LIKE [804] clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because “_” is a wildcard that matches any single character, you should escape it as “_” to match it literally. In practice, this is rarely necessary.

12.4.5.26 SHOW WARNINGS Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

SHOW WARNINGS shows the error, warning, and note messages that resulted from the last statement that generated messages in the current session. It shows nothing if the last statement used a table and generated no messages. (That is, a statement that uses a table but generates no messages clears the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.

Warnings are generated for DML statements such as INSERT, UPDATE, and LOAD DATA INFILE as well as DDL statements such as CREATE TABLE and ALTER TABLE.

SHOW WARNINGS is implemented as of MySQL 4.1.0. A related statement, SHOW ERRORS, shows only the errors. See Section 12.4.5.11, "SHOW ERRORS Syntax".

The SHOW COUNT(*) WARNINGS statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the warning_count [434] variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

The value of warning_count [434] might be greater than the number of messages displayed by SHOW WARNINGS if the max_error_count [419] system variable is set so low that not all messages are stored. An example shown later in this section demonstrates how this can happen.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 12.2.7, "SELECT Syntax".

The MySQL server sends back the total number of errors, warnings, and notes resulting from the last statement. If you are using the C API, this value can be obtained by calling mysql_warning_count(). See Section 17.6.6.70, "mysql_warning_count()".

Note that the framework for warnings was added in MySQL 4.1.0, at which point many statements did not generate warnings. In 4.1.1, the situation is much improved, with warnings generated for statements such as LOAD DATA INFILE and DML statements such as INSERT, UPDATE, CREATE TABLE, and ALTER TABLE.

The following DROP TABLE statement results in a note:

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;
+-------+------+-----------------------------+
| Level | Code | Message                     |
+-------+------+-----------------------------+
| Note  | 1051 | Unknown table 'no_such_table' |
+-------+------+-----------------------------+
```

Here is a simple example that shows a syntax warning for CREATE TABLE and conversion warnings for INSERT:

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
         'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'),(NULL,'test'),
    -> (300,'Open Source');
```

```
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 4

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
*************************** 4. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)
```

The maximum number of error, warning, and note messages to store is controlled by the `max_error_count` [419] system variable. By default, its value is 64. To change the number of messages you want stored, change the value of `max_error_count` [419]. In the following example, the `ALTER TABLE` statement produces three warning messages, but only one is stored because `max_error_count` [419] has been set to 1:

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> SET max_error_count=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)
```

To disable warnings, set `max_error_count` [419] to 0. In this case, `warning_count` [434] still indicates how many warnings have occurred, but none of the messages are stored.

As of MySQL 4.1.11, you can set the `sql_notes` [429] session variable to 0 to cause `Note`-level warnings not to be recorded.

12.4.6 Other Administrative Statements

12.4.6.1 CACHE INDEX Syntax

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It is used only for `MyISAM` tables. After the indexes have been assigned, they can be preloaded into the cache if desired with `LOAD INDEX INTO CACHE`.

The following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The syntax of `CACHE INDEX` enables you to specify that only particular indexes from a table should be assigned to the cache. The current implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters can be accessed as members of a structured system variable. See Section 5.1.4.1, “Structured System Variables”.

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it become assigned to the default key cache again.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

`CACHE INDEX` was added in MySQL 4.1.1.

12.4.6.2 FLUSH Syntax

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL]
    flush_option [, flush_option] ...
```

The FLUSH statement clears or reloads various internal caches used by MySQL. One variant acquires a lock. To execute FLUSH, you must have the RELOAD [492] privilege.

Before MySQL 4.1.1, FLUSH statements are not written to the binary log. As of MySQL 4.1.1, FLUSH statements are written to the binary log so that they will be replicated to replication slaves. Logging can be suppressed with the optional NO_WRITE_TO_BINLOG keyword or its alias LOCAL.

Note

FLUSH LOGS, FLUSH MASTER, FLUSH SLAVE, and FLUSH TABLES WITH READ LOCK are not written to the binary log in any case because they would cause problems if replicated to a slave.

The RESET statement is similar to FLUSH. See Section 12.4.6.5, "RESET Syntax", for information about using the RESET statement with replication.

flush_option can be any of the following items:

- DES_KEY_FILE

  Reloads the DES keys from the file that was specified with the --des-key-file [386] option at server startup time.

- HOSTS

  Empties the host cache tables. You should flush the host tables if some of your hosts change IP address or if you get the error message Host '*host_name*' is blocked. When more than max_connect_errors [419] errors occur successively for a given host while connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connection requests. Flushing the host tables enables further connection attempts from the host. See Section B.5.2.6, "Host '*host_name*' is blocked". You can start mysqld with --max_connect_errors=999999999 [419] to avoid this error message.

- LOGS

  Closes and reopens all log files. If you have specified an update log file or a binary log file without an extension, the extension number of the log file is incremented by one relative to the previous file. If you have used an extension in the file name, MySQL closes and reopens the update log or binary log file. See Section 5.3.4, "The Binary Log". On Unix, this is the same thing as sending a SIGHUP signal to the mysqld server (except on some Mac OS X 10.3 versions where mysqld ignores SIGHUP and SIGQUIT).

  Beginning with MySQL 4.0.10, if the server was started with the --log-error [388] option), error log file renaming occurs as described in Section 5.3.1, "The Error Log".

- MASTER

  Deletes all binary logs, resets the binary log index file and creates a new binary log. FLUSH MASTER is deprecated in favor of RESET MASTER, and is supported for backward compatibility only. See Section 12.5.1.2, "RESET MASTER Syntax".

- PRIVILEGES

  Reloads the privileges from the grant tables in the mysql database. On Unix, this also occurs if the server receives a SIGHUP signal.

  The server caches information in memory as a result of GRANT statements. This memory is not released by the corresponding REVOKE statements, so for a server that executes many instances of the

statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.

- `QUERY CACHE`

  Defragment the query cache to better utilize its memory. `FLUSH QUERY CACHE` does not remove any queries from the cache, unlike `FLUSH TABLES` or `RESET QUERY CACHE`.

- `SLAVE`

  Resets all replication slave parameters, including relay log files and replication position in the master's binary logs. `FLUSH SLAVE` is deprecated in favor of `RESET SLAVE`, and is supported for backward compatibility only. See Section 12.5.2.5, “`RESET SLAVE` Syntax”.

- `STATUS`

  Resets most status variables to zero. This is something you should use only when debugging a query. See Section 1.8, “How to Report Bugs or Problems”.

- `TABLES`

  `FLUSH TABLES` has several variant forms. `FLUSH TABLE` is a synonym for `FLUSH TABLES`, except that `TABLE` does not work with the `WITH READ LOCK` variant.

  - `FLUSH TABLES`

    Closes all open tables, forces all tables in use to be closed, and flushes the query cache. `FLUSH TABLES` also removes all query results from the query cache, like the `RESET QUERY CACHE` statement.

  - `FLUSH TABLES` *`tbl_name`* `[,` *`tbl_name`*`] ...`

    With a list of one or more comma-separated table names, this is like `FLUSH TABLES` with no names except that the server flushes only the named tables. No error occurs if a named table does not exist.

  - `FLUSH TABLES WITH READ LOCK`

    Closes all open tables and locks all tables for all databases with a global read lock until you explicitly release the lock by executing `UNLOCK TABLES`. This is a very convenient way to get backups if you have a file system such as Veritas or ZFS that can take snapshots in time.

    `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits:

    - `UNLOCK TABLES` implicitly commits any active transaction only if any tables currently have been locked with `LOCK TABLES`. The commit does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table locks.

    - Beginning a transaction causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

- `USER_RESOURCES`

  Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. `FLUSH USER_RESOURCES` does not

apply to the limit on maximum simultaneous connections. See Section 5.6.4, “Setting Account Resource Limits”.

The `mysqladmin` utility provides a command-line interface to some flush operations, using commands such as `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, and `flush-tables`.

12.4.6.3 `KILL` Syntax

```
KILL thread_id
```

Each connection to `mysqld` runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement.

If you have the `PROCESS` [492] privilege, you can see all threads. If you have the `SUPER` [493] privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

You can also use the `mysqladmin processlist` and `mysqladmin kill` commands to examine and kill threads.

Note

You cannot use `KILL` with the Embedded MySQL Server library because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

When you use `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die because the kill flag is checked only at specific intervals:

- In `SELECT`, `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- During `ALTER TABLE`, the kill flag is checked before each block of rows are read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.
- During `UPDATE` or `DELETE` operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. Note that if you are not using transactions, the changes are not rolled back.
- `GET_LOCK()` [877] aborts and returns `NULL`.
- An `INSERT DELAYED` thread quickly flushes (inserts) all rows it has in memory and then terminates.
- If the thread is in the table lock handler (state: `Locked`), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.
- Some threads might refuse to be killed. For example, `REPAIR TABLE`, `CHECK TABLE`, and `OPTIMIZE TABLE` cannot be killed before MySQL 4.1 and run to completion. This is changed: `REPAIR TABLE` and `OPTIMIZE TABLE` can be killed as of MySQL 4.1.0, as can `CHECK TABLE` as of MySQL 4.1.3. However, killing a `REPAIR TABLE` or `OPTIMIZE TABLE` operation on a `MyISAM` table results in a table that *is* corrupted and is unusable (reads and writes to it fail) until you optimize or repair it again (without interruption).
- If `CHECK TABLE` finds a problem for an `InnoDB` table, the server shuts down to prevent error propagation. Details of the error will be written to the error log.

12.4.6.4 LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]
```

The LOAD INDEX INTO CACHE statement preloads a table index into the key cache to which it has been assigned by an explicit CACHE INDEX statement, or into the default key cache otherwise. LOAD INDEX INTO CACHE is used only for MyISAM tables.

The IGNORE LEAVES modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

The following statement preloads nodes (index blocks) of indexes for the tables t1 and t2:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

This statement preloads all index blocks from t1. It preloads only blocks for the nonleaf nodes from t2.

The syntax of LOAD INDEX INTO CACHE enables you to specify that only particular indexes from a table should be preloaded. The current implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

LOAD INDEX INTO CACHE fails unless all indexes in a table have the same block size. You can determine index block sizes for a table by using myisamchk -dv and checking the Blocksize column.

LOAD INDEX INTO CACHE was added in MySQL 4.1.1.

12.4.6.5 RESET Syntax

```
RESET reset_option [, reset_option] ...
```

The RESET statement is used to clear the state of various server operations. You must have the RELOAD [492] privilege to execute RESET.

RESET acts as a stronger version of the FLUSH statement. See Section 12.4.6.2, “FLUSH Syntax”.

reset_option can be any of the following:

- MASTER

  Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file. Previously named FLUSH MASTER. See Section 12.5.1, “SQL Statements for Controlling Master Servers”.

- QUERY CACHE

  Removes all query results from the query cache.

- SLAVE

  Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one. Previously named FLUSH SLAVE. See Section 12.5.2, “SQL Statements for Controlling Slave Servers”.

12.5 Replication Statements

Replication can be controlled through the SQL interface using the statements described in this section. One group of statements controls master servers, the other controls slave servers.

12.5.1 SQL Statements for Controlling Master Servers

This section discusses statements for managing master replication servers. Section 12.5.2, “SQL Statements for Controlling Slave Servers”, discusses statements for managing slave servers.

In addition to the statements described here, the following SHOW statements are used with master servers in replication. For information about these statements, see Section 12.4.5, “SHOW Syntax”.

- SHOW BINARY LOGS
- SHOW BINLOG EVENTS
- SHOW MASTER STATUS
- SHOW SLAVE HOSTS

12.5.1.1 PURGE BINARY LOGS Syntax

```
PURGE { BINARY | MASTER } LOGS
    { TO 'log_name' | BEFORE datetime_expr }
```

The binary log is a set of files that contain information about data modifications made by the MySQL server. The log consists of a set of binary log files, plus an index file (see Section 5.3.4, “The Binary Log”).

The PURGE BINARY LOGS statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. BINARY and MASTER are synonyms, but only MASTER can be used before MySQL 4.1.1. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

This statement has no effect if the server was not started with the --log-bin [1181] option to enable binary logging.

Examples:

```
PURGE BINARY LOGS TO 'mysql-bin.010';
PURGE BINARY LOGS BEFORE '2008-04-02 22:46:26';
```

The BEFORE variant's *datetime_expr* argument should evaluate to a DATETIME value (a value in 'YYYY-MM-DD hh:mm:ss' format).

This statement is safe to run while slaves are replicating. You need not stop them. If you have an active slave that currently is reading one of the log files you are trying to delete, this statement does nothing and fails with an error. However, if a slave is not connected and you happen to purge one of the log files it has yet to read, the slave will be unable to replicate after it reconnects.

To safely purge binary log files, follow this procedure:

1. On each slave server, use `SHOW SLAVE STATUS` to check which log file it is reading.
2. Obtain a listing of the binary log files on the master server with `SHOW BINARY LOGS`.
3. Determine the earliest log file among all the slaves. This is the target file. If all the slaves are up to date, this is the last log file on the list.
4. Make a backup of all the log files you are about to delete. (This step is optional, but always advisable.)
5. Purge all log files up to but not including the target file.

You can also set the `expire_logs_days` [411] system variable to expire binary log files automatically after a given number of days (see Section 5.1.3, “Server System Variables”). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master.

12.5.1.2 `RESET MASTER` Syntax

```
RESET MASTER
```

Deletes all binary log files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file. This statement is intended to be used only when the master is started for the first time.

This statement was named `FLUSH MASTER` before MySQL 3.23.26.

Important

The effects of `RESET MASTER` differ from those of `PURGE BINARY LOGS` in 2 key ways:

1. `RESET MASTER` removes *all* binary log files that are listed in the index file, leaving only a single, empty binary log file with a numeric suffix of `.000001`, whereas the numbering is not reset by `PURGE BINARY LOGS`.
2. `RESET MASTER` is *not* intended to be used while any replication slaves are running. The behavior of `RESET MASTER` when used while slaves are running is undefined (and thus unsupported), whereas `PURGE BINARY LOGS` may be safely used while replication slaves are running.

See also Section 12.5.1.1, “`PURGE BINARY LOGS` Syntax”.

`RESET MASTER` can prove useful when you first set up the master and the slave, so that you can verify the setup as follows:

1. Start the master and slave, and start replication (see Section 14.4, “How to Set Up Replication”).
2. Execute a few test queries on the master.
3. Check that the queries were replicated to the slave.
4. When replication is running correctly, issue `STOP SLAVE` followed by `RESET SLAVE` on the slave, then verify that any unwanted data no longer exists on the slave.
5. Issue `RESET MASTER` on the master to clean up the test queries.

After verifying the setup and getting rid of any unwanted and log files generated by testing, you can start the slave and begin replicating.

12.5.1.3 SET sql_log_bin Syntax

```
SET sql_log_bin = {0|1}
```

Disables or enables binary logging for the current session (sql_log_bin [429] is a session variable) if the client that has the SUPER [493] privilege. The statement fails with an error if the client does not have that privilege. (Before MySQL 4.1.2, the statement was simply ignored in that case.)

12.5.2 SQL Statements for Controlling Slave Servers

This section discusses statements for managing slave replication servers. Section 12.5.1, "SQL Statements for Controlling Master Servers", discusses statements for managing master servers.

In addition to the statements described here, SHOW SLAVE STATUS is also used with replication slaves. For information about this statement, see Section 12.4.5.21, "SHOW SLAVE STATUS Syntax".

12.5.2.1 CHANGE MASTER TO Syntax

```
CHANGE MASTER TO option [, option] ...

option:
    MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = interval
  | MASTER_LOG_FILE = 'master_log_name'
  | MASTER_LOG_POS = master_log_pos
  | RELAY_LOG_FILE = 'relay_log_name'
  | RELAY_LOG_POS = relay_log_pos
  | MASTER_SSL = {0|1}
  | MASTER_SSL_CA = 'ca_file_name'
  | MASTER_SSL_CAPATH = 'ca_directory_name'
  | MASTER_SSL_CERT = 'cert_file_name'
  | MASTER_SSL_KEY = 'key_file_name'
  | MASTER_SSL_CIPHER = 'cipher_list'
```

CHANGE MASTER TO changes the parameters that the slave server uses for connecting to the master server, for reading the master binary log, and reading the slave relay log. It also updates the contents of the master.info and relay-log.info files. To use CHANGE MASTER TO, the slave replication threads must be stopped (use STOP SLAVE if necessary).

Options not specified retain their value, except as indicated in the following discussion. Thus, in most cases, there is no need to specify options that do not change. For example, if the password to connect to your MySQL master has changed, you just need to issue these statements to tell the slave about the new password:

```
STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication
```

MASTER_HOST, MASTER_USER, MASTER_PASSWORD, and MASTER_PORT provide information to the slave about how to connect to its master:

- MASTER_HOST and MASTER_PORT are the host name (or IP address) of the master host and its TCP/IP port.

Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

If you specify the `MASTER_HOST` or `MASTER_PORT` option, the slave assumes that the master server is different from before (even if the option value is the same as its current value.) In this case, the old values for the master binary log file name and position are considered no longer applicable, so if you do not specify `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the statement, `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4` are silently appended to it.

Setting `MASTER_HOST=''`—that is, setting its value explicitly to an empty string—is *not* the same as not setting it at all. Setting this option to an empty string causes `START SLAVE` subsequently to fail. (Bug #28796)

- `MASTER_USER` and `MASTER_PASSWORD` are the user name and password of the account to use for connecting to the master.

The `MASTER_SSL_xxx` options provide information about using SSL for the connection. They correspond to the `--ssl-xxx` options described in Section 5.6.6.3, “SSL Command Options”. These options are available beginning with MySQL 4.1.1. They can be changed even on slaves that are compiled without SSL support. They are saved to the `master.info` file, but are ignored if the slave does not have SSL support enabled.

`MASTER_CONNECT_RETRY` specifies how many seconds to wait between connect retries. The default is 60. The *number* of reconnection attempts is limited by the `--master-retry-count` [1174] server option; for more information, see Section 14.8, “Replication and Binary Logging Options and Variables”.

The relay log options (`RELAY_LOG_FILE` and `RELAY_LOG_POS`) are available beginning with MySQL 4.0.

`MASTER_LOG_FILE` and `MASTER_LOG_POS` are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. `RELAY_LOG_FILE` and `RELAY_LOG_POS` are the coordinates at which the slave SQL thread should begin reading from the relay log the next time the thread starts. If you specify either of `MASTER_LOG_FILE` or `MASTER_LOG_POS`, you cannot specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. If neither of `MASTER_LOG_FILE` or `MASTER_LOG_POS` is specified, the slave uses the last coordinates of the *slave SQL thread* before `CHANGE MASTER TO` was issued. This ensures that replication has no discontinuity, even if the slave SQL thread was late compared to the slave I/O thread, when you just want to change, say, the password to use. This safe behavior was introduced starting from MySQL 4.0.17 and 4.1.1. (Before these versions, the coordinates used were the last coordinates of the slave I/O thread before `CHANGE MASTER TO` was issued. This caused the SQL thread to possibly lose some events from the master, thus breaking replication.)

`CHANGE MASTER TO` *deletes all relay log files* and starts a new one, unless you specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. In that case, relay log files are kept; as of MySQL 4.1.1, the `relay_log_purge` [426] global variable is set silently to 0.

`CHANGE MASTER TO` is useful for setting up a slave when you have the snapshot of the master and have recorded the master binary log coordinates corresponding to the time of the snapshot. After loading the snapshot into the slave to synchronize it to the slave, you can run `CHANGE MASTER TO MASTER_LOG_FILE='log_name', MASTER_LOG_POS=log_pos` on the slave to specify the coodinates at which the slave should begin reading the master binary log.

The following example changes the master server the slave uses and establishes the master binary log coordinates from which the slave begins reading. This is used when you want to set up the slave to replicate the master:

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='bigs3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the slave has relay log files that you want it to execute again for some reason. To do this, the master need not be reachable. You need only use `CHANGE MASTER TO` and start the SQL thread (`START SLAVE SQL_THREAD`):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

You can even use the second operation in a nonreplication setup with a standalone, nonslave server for recovery following a crash. Suppose that your server has crashed and you have restored it from a backup. You want to replay the server's own binary log files (not relay log files, but regular binary log files), named (for example) `myhost-bin.*`. First, make a backup copy of these binary log files in some safe place, in case you don't exactly follow the procedure below and accidentally have the server purge the binary log. If using MySQL 4.1.1 or newer, use `SET GLOBAL relay_log_purge=0` for additional safety. Then start the server without the `--log-bin` [1181] option. Before MySQL 4.0.19, start it with a new server ID; in newer versions there is no need; simply use the `--replicate-same-server-id` [1177] option. Start it with `--relay-log=myhost-bin` [1175] (to make the server believe that these regular binary log files are relay log files) and `--skip-slave-start` [1179] options. After the server starts, issue these statements:

```
CHANGE MASTER TO
  RELAY_LOG_FILE='myhost-bin.153',
  RELAY_LOG_POS=410,
  MASTER_HOST='some_dummy_string';
START SLAVE SQL_THREAD;
```

The server reads and executes its own binary log files, thus achieving crash recovery. Once the recovery is finished, run `STOP SLAVE`, shut down the server, delete the `master.info` and `relay-log.info` files, and restart the server with its original options.

Specifying the `MASTER_HOST` option (even with a dummy value) is required to make the server think it is a slave.

12.5.2.2 `LOAD DATA FROM MASTER` Syntax

```
LOAD DATA FROM MASTER
```

Note

This feature is deprecated and should be avoided. It is subject to removal in a future version of MySQL.

Since the current implementation of `LOAD DATA FROM MASTER` and `LOAD TABLE FROM MASTER` is very limited, these statements are deprecated as of MySQL 4.1 and removed in MySQL 5.5.

The recommended alternative solution to using `LOAD DATA FROM MASTER` or `LOAD TABLE FROM MASTER` is using `mysqldump` or `mysqlhotcopy`. The latter requires Perl and two Perl modules (`DBI` and

DBD:mysql) and works for MyISAM and ARCHIVE tables only. With mysqldump, you can create SQL dumps on the master and pipe (or copy) these to a mysql client on the slave. This has the advantage of working for all storage engines, but can be quite slow, since it works using SELECT.

This statement takes a snapshot of the master and copies it to the slave. It updates the values of MASTER_LOG_FILE and MASTER_LOG_POS so that the slave starts replicating from the correct position. Any table and database exclusion rules specified with the --replicate-*-do-* and --replicate-*-ignore-* options are honored. --replicate-rewrite-db [1177] is *not* taken into account because a user could use this option to set up a nonunique mapping such as --replicate-rewrite-db="db1->db3" [1177] and --replicate-rewrite-db="db2->db3" [1177], which would confuse the slave when loading tables from the master.

Use of this statement is subject to the following conditions:

- It works only for MyISAM tables. Attempting to load a non-MyISAM table results in the following error:

```
ERROR 1189 (08S01): Net error reading from master
```

- It acquires a global read lock on the master while taking the snapshot, which prevents updates on the master during the load operation.

If you are loading large tables, you might have to increase the values of net_read_timeout [422] and net_write_timeout [422] on both the master and slave servers. See Section 5.1.3, “Server System Variables”.

Note that LOAD DATA FROM MASTER does *not* copy any tables from the mysql database. This makes it easy to have different users and privileges on the master and the slave.

To use LOAD DATA FROM MASTER, the replication account that is used to connect to the master must have the RELOAD [492] and SUPER [493] privileges on the master and the SELECT [492] privilege for all master tables you want to load. All master tables for which the user does not have the SELECT [492] privilege are ignored by LOAD DATA FROM MASTER. This is because the master hides them from the user: LOAD DATA FROM MASTER calls SHOW DATABASES to know the master databases to load, but SHOW DATABASES returns only databases for which the user has some privilege. See Section 12.4.5.8, “SHOW DATABASES Syntax”. On the slave side, the user that issues LOAD DATA FROM MASTER must have privileges for dropping and creating the databases and tables that are copied.

12.5.2.3 LOAD TABLE *tbl_name* FROM MASTER Syntax

```
LOAD TABLE tbl_name FROM MASTER
```

Note

This feature is deprecated and should be avoided. It is subject to removal in a future version of MySQL.

Since the current implementation of LOAD DATA FROM MASTER and LOAD TABLE FROM MASTER is very limited, these statements are deprecated as of MySQL 4.1 and removed in MySQL 5.5.

The recommended alternative solution to using LOAD DATA FROM MASTER or LOAD TABLE FROM MASTER is using mysqldump or mysqlhotcopy. The latter requires Perl and two Perl modules (DBI and DBD:mysql) and works for MyISAM and ARCHIVE tables only. With mysqldump, you can create SQL dumps on the master and pipe (or copy) these to a mysql client on the slave. This has the advantage of working for all storage engines, but can be quite slow, since it works using SELECT.

Transfers a copy of the table from the master to the slave. This statement is implemented mainly debugging `LOAD DATA FROM MASTER` operations. To use `LOAD TABLE`, the account used for connecting to the master server must have the `RELOAD` [492] and `SUPER` [493] privileges on the master and the `SELECT` [492] privilege for the master table to load. On the slave side, the user that issues `LOAD TABLE FROM MASTER` must have privileges for dropping and creating the table.

The conditions for `LOAD DATA FROM MASTER` apply here as well. For example, `LOAD TABLE FROM MASTER` works only for `MyISAM` tables. The timeout notes for `LOAD DATA FROM MASTER` apply as well.

12.5.2.4 `MASTER_POS_WAIT()` Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos [, timeout])
```

This is actually a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See Section 11.14, “Miscellaneous Functions”, for a full description.

12.5.2.5 `RESET SLAVE` Syntax

```
RESET SLAVE
```

`RESET SLAVE` makes the slave forget its replication position in the master's binary log. This statement is meant to be used for a clean start: It deletes the `master.info` and `relay-log.info` files, all the relay log files, and starts a new relay log file. To use `RESET SLAVE`, the slave replication threads must be stopped (use `STOP SLAVE` if necessary).

Note

All relay log files are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a `STOP SLAVE` statement or if the slave is highly loaded.)

Connection information stored in the `master.info` file is immediately reset using any values specified in the corresponding startup options. This information includes values such as master host, master port, master user, and master password. If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and `RESET SLAVE` is issued, these replicated temporary tables are deleted on the slave.

This statement was named `FLUSH SLAVE` before MySQL 3.23.26.

12.5.2.6 `SET GLOBAL SQL_SLAVE_SKIP_COUNTER` Syntax

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N
```

This statement skips the next *`N`* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave threads are not running. Otherwise, it produces an error.

Before MySQL 4.0, omit the `GLOBAL` keyword from the statement.

When using this statement, it is important to understand that the binary log is actually organized as a sequence of groups known as *event groups*. Each event group consists of a sequence of events.

- For transactional tables, an event group corresponds to a transaction.
- For nontransactional tables, an event group corresponds to a single SQL statement.

Note

A single transaction can contain changes to both transactional and nontransactional tables.

When you use `SET [GLOBAL] SQL_SLAVE_SKIP_COUNTER` to skip events and the result is in the middle of a group, the slave continues to skip events until it reaches the end of the group. Execution then starts with the next event group.

12.5.2.7 START SLAVE Syntax

```
START SLAVE [thread_type [, thread_type] ... ]
START SLAVE [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
START SLAVE [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos

thread_type: IO_THREAD | SQL_THREAD
```

`START SLAVE` with no *`thread_type`* options starts both of the slave threads. The I/O thread reads events from the master server and stores them in the relay log. The SQL thread reads events from the relay log and executes them. `START SLAVE` requires the `SUPER` [493] privilege.

If `START SLAVE` succeeds in starting the slave threads, it returns without any error. However, even in that case, it might be that the slave threads start and then later stop (for example, because they do not manage to connect to the master or read its binary log, or some other problem). `START SLAVE` does not warn you about this. You must check the slave's error log for error messages generated by the slave threads, or check that they are running satisfactorily with `SHOW SLAVE STATUS`.

`START SLAVE` sends an acknowledgment to the user after both the I/O thread and the SQL thread have started. However, the I/O thread may not yet have connected. For this reason, a successful `START SLAVE` causes `SHOW SLAVE STATUS` to show `Slave_SQL_Running=Yes`, but this does not guarantee that `Slave_IO_Running=Yes` (because `Slave_IO_Running=Yes` only if the I/O thread is running *and connected*). For more information, see Section 12.4.5.21, “SHOW SLAVE STATUS Syntax”.

As of MySQL 4.0.2, you can add `IO_THREAD` and `SQL_THREAD` options to the statement to name which of the threads to start.

As of MySQL 4.1.1, an `UNTIL` clause may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary log or in the slave relay log. When the SQL thread reaches that point, it stops. If the `SQL_THREAD` option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the `UNTIL` clause is ignored and a warning is issued.

For an `UNTIL` clause, you must specify both a log file name and position. Do not mix master and relay log options.

Any `UNTIL` condition is reset by a subsequent `STOP SLAVE` statement, a `START SLAVE` statement that includes no `UNTIL` clause, or a server restart.

The `UNTIL` clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate an event. For example, if an unwise `DROP TABLE` statement was executed on the master, you can use `UNTIL` to tell the slave to execute up to

that point but no farther. To find what the event is, use `mysqlbinlog` with the master binary log or slave relay log, or by using a `SHOW BINLOG EVENTS` statement.

If you are using `UNTIL` to have the slave process replicated queries in sections, it is recommended that you start the slave with the `--skip-slave-start` [1179] option to prevent the SQL thread from running when the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The `SHOW SLAVE STATUS` statement includes output fields that display the current values of the `UNTIL` condition.

This statement is called `SLAVE START` before MySQL 4.0.5. `SLAVE START` is still accepted for backward compatibility, but is now deprecated.

12.5.2.8 `STOP SLAVE` Syntax

```
STOP SLAVE [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

Stops the slave threads. `STOP SLAVE` requires the `SUPER` [493] privilege.

Like `START SLAVE`, as of MySQL 4.0.2, this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to be stopped.

This statement is called `SLAVE STOP` before MySQL 4.0.5. `SLAVE STOP` is still accepted for backward compatibility, but is deprecated.

12.6 SQL Syntax for Prepared Statements

Support for server-side prepared statements was added in MySQL 4.1. This support takes advantage of the efficient client/server binary protocol, provided that you use an appropriate client programming interface. Candidate interfaces include the MySQL C API client library (for C programs), MySQL Connector/J (for Java programs), and MySQL Connector/Net. For example, the C API provides a set of function calls that make up its prepared statement API. See Section 17.6.7, “C API Prepared Statements”. Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the `mysqli` extension, available in PHP 5.0 and later.

Beginning with MySQL 4.1.3, an alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that enables you to send SQL statements to the server to be executed, such as the `mysql` client program.
- You can use it even if the client is using an old version of the client library. The only requirement is that you be able to connect to a server that is recent enough to support SQL syntax for prepared statements.

SQL syntax for prepared statements is intended to be used for situations such as these:

- You want to test how prepared statements work in your application before coding it.
- An application has problems executing prepared statements and you want to determine interactively what the problem is.

- You want to create a test case that describes a problem you are having with prepared statements, so that you can file a bug report.
- You need to use prepared statements but do not have access to a programming API that supports them.

SQL syntax for prepared statements is based on three SQL statements:

- `PREPARE` prepares a statement for execution (see Section 12.6.1, “`PREPARE` Syntax”).
- `EXECUTE` executes a prepared statement (see Section 12.6.2, “`EXECUTE` Syntax”).
- `DEALLOCATE PREPARE` releases a prepared statement (see Section 12.6.3, “`DEALLOCATE PREPARE` Syntax”).

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|          5 |
+------------+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
mysql> DEALLOCATE PREPARE stmt2;
```

Here is an additional example which demonstrates how to choose the table on which to perform a query at runtime, by storing the name of the table as a user variable:

```
mysql> USE test;
mysql> CREATE TABLE t1 (a INT NOT NULL);
mysql> INSERT INTO t1 VALUES (4), (8), (11), (32), (80);

mysql> SET @table = 't1';
mysql> SET @s = CONCAT('SELECT * FROM ', @table);

mysql> PREPARE stmt3 FROM @s;
mysql> EXECUTE stmt3;
+----+
| a  |
+----+
```

```
|  4 |
|  8 |
| 11 |
| 32 |
| 80 |
+----+

mysql> DEALLOCATE PREPARE stmt3;
```

A prepared statement is specific to the session in which it was created. If you terminate a session without deallocating a previously prepared statement, the server deallocates it automatically.

To guard against too many prepared statements being created simultaneously, set the `max_prepared_stmt_count` [420] system variable. To prevent the use of prepared statements, set the value to 0.

The following SQL statements can be used in prepared statements:

```
ALTER TABLE
COMMIT
{CREATE | DROP} INDEX
{CREATE | DROP} TABLE
DELETE
DO
INSERT
RENAME TABLE
REPLACE
SELECT
SET
SHOW (most variants)
UPDATE
```

Other statements are not supported in MySQL 4.1.

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to `PREPARE` cannot itself be a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the `mysql_stmt_prepare()` C API function to prepare a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by "`;`" characters).

12.6.1 PREPARE Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

The `PREPARE` statement prepares a SQL statement and assigns it a name, *`stmt_name`*, by which to refer to the statement later. The prepared statement is executed with `EXECUTE` and released with `DEALLOCATE PREPARE`. For examples, see Section 12.6, "SQL Syntax for Prepared Statements".

Statement names are not case sensitive. *`preparable_stmt`* is either a string literal or a user variable that contains the text of the SQL statement. The text must represent a single statement, not multiple statements. Within the statement, `?` characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The `?` characters should not be enclosed within quotation marks, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

The scope of a prepared statement is the session within which it is created, which as several implications:

- A prepared statement created in one session is not available to other sessions.
- When a session ends, whether normally or abnormally, its prepared statements no longer exist. If auto-reconnect is enabled, the client is not notified that the connection was lost. For this reason, clients may wish to disable auto-reconnect. See Section 17.6.14, “Controlling Automatic Reconnection Behavior”.

12.6.2 EXECUTE Syntax

```
EXECUTE stmt_name
    [USING @var_name [, @var_name] ...]
```

After preparing a statement with PREPARE, you execute it with an EXECUTE statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a USING clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the USING clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

For examples, see Section 12.6, “SQL Syntax for Prepared Statements”.

12.6.3 DEALLOCATE PREPARE Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

To deallocate a prepared statement produced with PREPARE, use a DEALLOCATE PREPARE statement that refers to the prepared statement name. Attempting to execute a prepared statement after deallocating it results in an error.

For examples, see Section 12.6, “SQL Syntax for Prepared Statements”.

12.7 MySQL Utility Statements

12.7.1 DESCRIBE Syntax

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

DESCRIBE provides information about the columns in a table. It is a shortcut for SHOW COLUMNS FROM. (See Section 12.4.5.5, “SHOW COLUMNS Syntax”.)

col_name can be a column name, or a string containing the SQL “%” and “_” wildcard characters to obtain output only for the columns with names matching the string. There is no need to enclose the string within quotation marks unless it contains spaces or other special characters.

```
mysql> DESCRIBE City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
```

```
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  |      | PRI | NULL    | auto_increment |
| Name       | char(35) |      |     |         |                |
| Country    | char(3)  |      | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  |      |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

The description for `SHOW COLUMNS` provides more information about the output columns (see Section 12.4.5.5, “SHOW COLUMNS Syntax”).

If the data types differ from what you expect them to be based on a `CREATE TABLE` statement, note that MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in Section 12.1.5.2, “Silent Column Specification Changes”.

The `DESCRIBE` statement is provided for compatibility with Oracle.

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables. See Section 12.4.5, “SHOW Syntax”.

12.7.2 EXPLAIN Syntax

```
EXPLAIN [EXTENDED] SELECT select_options
```

Or:

```
EXPLAIN tbl_name
```

The `EXPLAIN` statement can be used either as a synonym for `DESCRIBE` or as a way to obtain information about how MySQL executes a `SELECT` statement:

- When you precede a `SELECT` statement with the keyword `EXPLAIN`, MySQL displays information from the optimizer about the query execution plan. That is, MySQL explains how it would process the `SELECT`, including information about how tables are joined and in which order. As of MySQL 4.1, `EXPLAIN EXTENDED` can be used to provide additional information.

  For information on how to use `EXPLAIN` and `EXPLAIN EXTENDED` to obtain query execution plan information, see Section 7.2.1, “Optimizing Queries with EXPLAIN”.

- `EXPLAIN tbl_name` is synonymous with `DESCRIBE tbl_name` or `SHOW COLUMNS FROM tbl_name`.

  For a description of the `DESCRIBE` and `SHOW COLUMNS` statements, see Section 12.7.1, “DESCRIBE Syntax”, and Section 12.4.5.5, “SHOW COLUMNS Syntax”.

12.7.3 HELP Syntax

```
HELP 'search_string'
```

The `HELP` statement returns online information from the MySQL Reference manual. Its proper operation requires that the help tables in the `mysql` database be initialized with help topic information (see Section 5.1.7, “Server-Side Help”).

The `HELP` statement searches the help tables for the given search string and displays the result of the search. The search string is not case sensitive.

The HELP statement understands several types of search strings:

- At the most general level, use `contents` to retrieve a list of the top-level help categories:

```
HELP 'contents'
```

- For a list of topics in a given help category, such as `Data Types`, use the category name:

```
HELP 'data types'
```

- For help on a specific help topic, such as the `ASCII()` [793] function or the `CREATE TABLE` statement, use the associated keyword or keywords:

```
HELP 'ascii'
HELP 'create table'
```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string will return a list of items or the help information for a single help topic. However, you can tell what kind of response `HELP` returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the `mysql` client, but note that `mysql` itself reformats `HELP` result sets in a different way.

- Empty result set

  No match could be found for the search string.

- Result set containing a single row with three columns

  This means that the search string yielded a hit for the help topic. The result has three columns:

  - `name`: The topic name.
  - `description`: Descriptive help text for the topic.
  - `example`: Usage example or examples. This column might be blank.

  Example: `HELP 'replace'`

  Yields:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

- Result set containing multiple rows with two columns

  This means that the search string matched many help topics. The result set indicates the help topic names:

  - `name`: The help topic name.

- `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

Example: `HELP 'status'`

Yields:

```
+----------------------+----------------+
| name                 | is_it_category |
+----------------------+----------------+
| SHOW                 | N              |
| SHOW ENGINE          | N              |
| SHOW INNODB STATUS   | N              |
| SHOW MASTER STATUS   | N              |
| SHOW SLAVE STATUS    | N              |
| SHOW STATUS          | N              |
| SHOW TABLE STATUS    | N              |
+----------------------+----------------+
```

- Result set containing multiple rows with three columns

  This means the search string matches a category. The result set contains category entries:

  - `source_category_name`: The help category name.
  - `name`: The category or topic name
  - `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

  Example: `HELP 'functions'`

  Yields:

```
+---------------------+-------------------------+----------------+
| source_category_name | name                    | is_it_category |
+---------------------+-------------------------+----------------+
| Functions           | CREATE FUNCTION         | N              |
| Functions           | DROP FUNCTION           | N              |
| Functions           | Bit Functions           | Y              |
| Functions           | Comparison operators    | Y              |
| Functions           | Control flow functions  | Y              |
| Functions           | Date and Time Functions | Y              |
| Functions           | Encryption Functions    | Y              |
| Functions           | Information Functions   | Y              |
| Functions           | Logical operators       | Y              |
| Functions           | Miscellaneous Functions | Y              |
| Functions           | Numeric Functions       | Y              |
| Functions           | String Functions        | Y              |
+---------------------+-------------------------+----------------+
```

If you intend to use the `HELP` statement while other tables are locked with `LOCK TABLES`, you must also lock the required `mysql.help_xxx` tables.

The `HELP` statement was added in MySQL 4.1.

12.7.4 USE Syntax

```
USE db_name
```

The USE *db_name* statement tells MySQL to use the *db_name* database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another USE statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

Making a particular database the default by means of the USE statement does not preclude you from accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
  WHERE author.editor_id = db2.editor.editor_id;
```

The USE statement is provided for compatibility with Sybase.

Chapter 13 Storage Engines

Table of Contents

MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle nontransaction-safe tables:

- The original storage engine was ISAM, which managed nontransactional tables. This engine has been replaced by MyISAM and should no longer be used. It is deprecated in MySQL 4.1, and is removed in subsequent MySQL release series.
- In MySQL 3.23.0, the MyISAM and HEAP storage engines were introduced. MyISAM is an improved replacement for ISAM. The HEAP storage engine provides in-memory tables. The MERGE storage engine

was added in MySQL 3.23.25. It enables a collection of identical MyISAM tables to be handled as a single table. All three of these storage engines handle nontransactional tables, and all are included in MySQL by default. Note that the HEAP storage engine has been renamed the MEMORY engine.

- The InnoDB and BDB storage engines that handle transaction-safe tables were introduced in later versions of MySQL 3.23. Both are available in source distributions as of MySQL 3.23.34a. BDB is included in MySQL-Max binary distributions on those operating systems that support it. InnoDB also is included in MySQL-Max binary distributions for MySQL 3.23. Beginning with MySQL 4.0, InnoDB is included by default in all MySQL binary distributions. In source distributions, you can enable or disable either engine by configuring MySQL as you like.

- The EXAMPLE storage engine was added in MySQL 4.1.3. It is a “stub” engine that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them. The purpose of this engine is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

- NDBCLUSTER is the storage engine used by MySQL Cluster to implement tables that are partitioned over many computers. It is available in source code distributions as of MySQL 4.1.2 and binary distributions as of MySQL-Max 4.1.3.

  MySQL Cluster is covered in a separate chapter of this Manual. See Chapter 15, *MySQL Cluster*, for more information.

- The ARCHIVE storage engine was added in MySQL 4.1.3. It is used for storing large amounts of data without indexes in a very small footprint.

- The CSV storage engine was added in MySQL 4.1.4. This engine stores data in text files using comma-separated values format.

- The BLACKHOLE storage engine was added in MySQL 4.1.11. This engine accepts but does not store data and retrievals always return an empty set.

To determine which storage engines your server supports by using the SHOW ENGINES statement. The value in the Support column indicates whether an engine can be used. A value of YES, NO, or DEFAULT indicates that an engine is available, not available, or available and currently set as the default storage engine.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
 Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
*************************** 2. row ***************************
 Engine: HEAP
Support: YES
Comment: Alias for MEMORY
*************************** 3. row ***************************
 Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 4. row ***************************
 Engine: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
*************************** 5. row ***************************
 Engine: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
...
```

This chapter describes each of the MySQL storage engines except for NDBCLUSTER, which is covered in Chapter 15, *MySQL Cluster*.

For information about storage engine support offered in commercial MySQL Server binaries, see *MySQL Enterprise Server 5.1*, on the MySQL Web site. The storage engines available might depend on which edition of Enterprise Server you are using.

When you create a new table, you can specify which storage engine to use by adding an ENGINE or TYPE table option to the CREATE TABLE statement:

```
CREATE TABLE t (i INT) ENGINE = INNODB;
CREATE TABLE t (i INT) TYPE = MEMORY;
```

ENGINE is the preferred term, but cannot be used before MySQL 4.0.18. TYPE is available beginning with MySQL 3.23.0, the first version of MySQL for which multiple storage engines were available. TYPE is supported for backward compatibility but is deprecated.

If you omit the ENGINE or TYPE option, the default storage engine is used. Normally, this is MyISAM, but you can change it by using the --default-storage-engine [386] or --default-table-type [386] server startup option, or by setting the default-storage-engine or default-table-type option in the my.cnf configuration file.

You can set the default storage engine to be used during the current session by setting the storage_engine [430] or table_type [431] variable:

```
SET storage_engine=MYISAM;
SET table_type=BDB;
```

When MySQL is installed on Windows using the MySQL Configuration Wizard, the InnoDB storage engine can be selected as the default instead of MyISAM. See Section 2.3.4.6, “The Database Usage Dialog”.

To convert a table from one storage engine to another, use an ALTER TABLE statement that indicates the new engine:

```
ALTER TABLE t ENGINE = MYISAM;
ALTER TABLE t TYPE = BDB;
```

See Section 12.1.5, “CREATE TABLE Syntax”, and Section 12.1.2, “ALTER TABLE Syntax”.

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine, usually MyISAM). (Before MySQL, MyISAM is always used for unavailable storage engines.) type MyISAM. This behavior is convenient when you want to copy tables between MySQL servers that support different storage engines. (For example, in a replication setup, perhaps your master server supports transactional storage engines for increased safety, but the slave servers use only nontransactional storage engines for greater speed.)

This automatic substitution of the default storage engine for unavailable engines can be confusing for new MySQL users. In MySQL 4.1, a warning is generated when a storage engine is automatically changed.

For new tables, MySQL always creates an .frm file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the storage engine. The server creates the .frm file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage.

A database may contain tables of different types. That is, tables need not all be created with the same storage engine.

Transaction-safe tables (TSTs) have several advantages over nontransaction-safe tables (NTSTs):

- They are safer. Even if MySQL crashes or you get hardware problems, you can get your data back, either by automatic recovery or from a backup plus the transaction log.
- You can combine many statements and accept them all at the same time with the COMMIT statement (if autocommit is disabled).
- You can execute ROLLBACK to ignore your changes (if autocommit is disabled).
- If an update fails, all of your changes are reverted. (With nontransaction-safe tables, all changes that have taken place are permanent.)
- Transaction-safe storage engines can provide better concurrency for tables that get many updates concurrently with reads.

You can combine transaction-safe and nontransaction-safe tables in the same statements to get the best of both worlds. However, although MySQL supports several transaction-safe storage engines, for best results, you should not mix different storage engines within a transaction with autocommit disabled. For example, if you do this, changes to nontransaction-safe tables still are committed immediately and cannot be rolled back. For information about this and other problems that can occur in transactions that use mixed storage engines, see Section 12.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

Note that to use the InnoDB storage engine in MySQL 3.23, you must configure at least the innodb_data_file_path [1067] startup option. In 4.0 and up, InnoDB uses default configuration values if you specify none. See Section 13.2.3, “InnoDB Configuration”.

Nontransaction-safe tables have several advantages of their own, all of which occur because there is no transaction overhead:

- Much faster
- Lower disk space requirements
- Less memory required to perform updates

13.1 The MyISAM Storage Engine

MyISAM is the default storage engine as of MySQL 3.23. It is based on the ISAM storage engine but has many useful extensions.

Each MyISAM table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format. The data file has an .MYD (MYData) extension. The index file has an .MYI (MYIndex) extension.

To specify explicitly that you want a MyISAM table, indicate that with an ENGINE table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term from MySQL 4.0.18 on and TYPE is deprecated.

Normally, the ENGINE or TYPE option is unnecessary; MyISAM is the default storage engine unless the default has been changed. To ensure that MyISAM is used in situations where the default might have been changed, specify the storage engine explicitly.

You can check or repair MyISAM tables with the mysqlcheck client or myisamchk utility. You can also compress MyISAM tables with myisampack to take up much less space. See Section 4.5.3, “mysqlcheck

— A Table Maintenance Program”, Section 4.6.2, “myisamchk — MyISAM Table-Maintenance Utility”, and Section 4.6.4, “myisampack — Generate Compressed, Read-Only MyISAM Tables”.

The following characteristics of the MyISAM storage engine are improvements over the older ISAM engine:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar processors.

  There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to permit better index compression.

- Large files (up to 63-bit file length) are supported on file systems and operating systems that support large files.

- The maximum number of indexes per table is 64 (32 before MySQL 4.1.2). This can be changed by changing the source and recompiling. The maximum number of columns per index is 16.

- The maximum key length is 1000 bytes (500 before MySQL 4.1.2). This can be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.

- Index files are usually much smaller with MyISAM than with ISAM. This means that MyISAM normally uses less system resources than ISAM, but needs more CPU time when inserting data into a compressed index.

- When rows are inserted in sorted order (as when you are using an AUTO_INCREMENT column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.

- Internal handling of one AUTO_INCREMENT column per table is supported. MyISAM automatically updates this column for INSERT/UPDATE. This makes AUTO_INCREMENT columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted as they are with ISAM. (When an AUTO_INCREMENT column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The AUTO_INCREMENT value can be reset with ALTER TABLE or myisamchk.

- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.

- MyISAM supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can INSERT new rows into it at the same time that other threads are reading from the table. A free block can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See Section 7.6.3, “Concurrent Inserts”.

- You can put the data file and index file in different directories on different physical devices to get more speed with the DATA DIRECTORY and INDEX DIRECTORY table options to CREATE TABLE. See Section 12.1.5, “CREATE TABLE Syntax”.

- BLOB and TEXT columns can be indexed.

- `NULL` values are permitted in indexed columns. This takes 0-1 bytes per key.
- As of MySQL 4.1, each character column can have a different character set.
- There is a flag in the `MyISAM` index file that indicates whether the table was closed correctly. If `mysqld` is started with the `--myisam-recover` [390] option, `MyISAM` tables are automatically checked when opened, and are repaired if the table wasn't closed properly.
- `myisamchk` marks tables as checked if you run it with the `--update-state` [316] option. `myisamchk --fast` checks only those tables that don't have this mark.
- `myisamchk --analyze` stores statistics for portions of keys, not only for whole keys as in `ISAM`.
- `myisampack` can pack `BLOB` and `VARCHAR` columns; `pack_isam` cannot.

`MyISAM` also supports the following features, which MySQL will be able to use in the near future:

- Support for a true `VARCHAR` type; a `VARCHAR` column starts with a length stored in one or two bytes.
- Tables with `VARCHAR` columns may have fixed or dynamic row length.
- The sum of the lengths of the `VARCHAR` and `CHAR` columns in a table may be up to 64KB.
- Arbitrary length `UNIQUE` constraints.

Additional Resources

- A forum dedicated to the `MyISAM` storage engine is available at http://forums.mysql.com/list.php?21.

13.1.1 `MyISAM` Startup Options

The following options to `mysqld` can be used to change the behavior of `MyISAM` tables. For additional information, see Section 5.1.2, “Server Command Options”.

- `--myisam-recover=`*mode* [390]

  Set the mode for automatic recovery of crashed `MyISAM` tables.

- `--delay-key-write=ALL` [386]

  Don't flush key buffers between writes for any `MyISAM` table.

Note

If you do this, you should not access `MyISAM` tables from another program (such as from another MySQL server or with `myisamchk`) when the tables are in use. Doing so risks index corruption. Using `--external-locking` [387] does not eliminate this risk.

The following system variables affect the behavior of `MyISAM` tables. For additional information, see Section 5.1.3, “Server System Variables”.

- `bulk_insert_buffer_size` [408]

  The size of the tree cache used in bulk insert optimization.

Note

This is a limit *per thread*!

- `myisam_max_extra_sort_file_size` [421]

  Used to help MySQL to decide when to use the slow but safe key cache index creation method.

  Note

  This parameter is given in megabytes before MySQL 4.0.3, and in bytes as of 4.0.3.

- `myisam_max_sort_file_size` [421]

  The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. This variable was added in MySQL 3.23.37.

  Note

  The value is given in megabytes before 4.0.3 and in bytes thereafter.

- `myisam_sort_buffer_size` [421]

  Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start `mysqld` with the `--myisam-recover` [390] option. In this case, when the server opens a `MyISAM` table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `--myisam-recover` [390] option, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Note that if the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form *`tbl_name-datetime`*`.BAK`. You should have a `cron` script that automatically moves these files from the database directories to backup media.

13.1.2 Space Needed for Keys

MyISAM tables use B-tree indexes. You can roughly calculate the size for the index file as (key_length+4)/0.67, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot of trailing space or is a VARCHAR column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In MyISAM tables, you can also prefix compress numbers by specifying the PACK_KEYS=1 table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

13.1.3 MyISAM Table Storage Formats

MyISAM supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the myisampack utility (see Section 4.6.4, "myisampack — Generate Compressed, Read-Only MyISAM Tables").

When you use CREATE TABLE or ALTER TABLE for a table that has no BLOB or TEXT columns, you can force the table format to FIXED or DYNAMIC with the ROW_FORMAT table option.

See Section 12.1.5, "CREATE TABLE Syntax", for information about ROW_FORMAT.

You can decompress (unpack) compressed MyISAM tables using myisamchk --unpack [318]; see Section 4.6.2, "myisamchk — MyISAM Table-Maintenance Utility", for more information.

13.1.3.1 Static (Fixed-Length) Table Characteristics

Static format is the default for MyISAM tables. It is used when the table contains no variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT). Each row is stored using a fixed number of bytes.

Of the three MyISAM storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format MyISAM file. In this case, myisamchk can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. Note that MyISAM table indexes can always be reconstructed based on the data rows.

Static-format tables have these characteristics:

- CHAR and BINARY columns are space-padded to the column width. This is also true for NUMERIC and DECIMAL columns.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use OPTIMIZE TABLE or myisamchk -r.
- Usually require more disk space than dynamic-format tables.

13.1.3.2 Dynamic Table Characteristics

Dynamic storage format is used if a MyISAM table contains any variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT), or if the table was created with the ROW_FORMAT=DYNAMIC table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in noncontiguous pieces) when it is made longer as a result of an update.

You can use OPTIMIZE TABLE or myisamchk -r to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.
- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). Note that this does not include columns that contain NULL values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Nonempty strings are saved as a length byte plus the string contents.
- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run OPTIMIZE TABLE or myisamchk -r from time to time to improve performance. Use myisamchk -ei to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8
```

  There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using myisamchk -ed. All links may be removed with OPTIMIZE TABLE or myisamchk -r.

13.1.3.3 Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the myisampack tool.

All MySQL distributions as of version 3.23.19 include myisampack by default. (This version is when MySQL was placed under the GPL.) For earlier versions, myisampack was included only with licenses or support agreements, but the server still can read tables that were compressed with myisampack. Compressed tables can be uncompressed with myisamchk. (For the ISAM storage engine, compressed tables can be created with pack_isam and uncompressed with isamchk.)

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).
- Each row is compressed separately, so there is very little access overhead. The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:
  - Suffix space compression.
  - Prefix space compression.
  - Numbers with a value of zero are stored using one bit.
  - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.
  - If a column has only a small set of possible values, the data type is converted to `ENUM`.
  - A column may use any combination of the preceding compression types.
- Can be used for fixed-length or dynamic-length rows.

Note

While a compressed table is read only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use `DROP` to drop the table, and `TRUNCATE TABLE` to empty the table.

13.1.4 MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

13.1.4.1 Corrupted MyISAM Tables

Even though the `MyISAM` table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if any of the following events occur:

- The `mysqld` process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as `myisamchk`) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or `MyISAM` code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:

```
Incorrect key file for table: '...'. Try to repair it
```

- Queries don't find rows in the table or return incomplete results.

You can check the health of a MyISAM table using the CHECK TABLE statement, and repair a corrupted MyISAM table with REPAIR TABLE. When mysqld is not running, you can also check or repair a table with the myisamchk command. See Section 12.4.2.3, "CHECK TABLE Syntax", Section 12.4.2.6, "REPAIR TABLE Syntax", and Section 4.6.2, "myisamchk — MyISAM Table-Maintenance Utility".

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent restarted mysqld message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See Section B.5.4.2, "What to Do If MySQL Keeps Crashing", and Section 18.4, "Porting to Other Systems".

13.1.4.2 Problems from Tables Not Being Closed Properly

Each MyISAM index file (.MYI file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from CHECK TABLE or myisamchk, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a FLUSH TABLES operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A MyISAM table is copied without first issuing LOCK TABLES and FLUSH TABLES.
- MySQL has crashed between an update and the final close. (Note that the table may still be okay, because MySQL always issues writes for everything between each statement.)
- A table was modified by myisamchk --recover or myisamchk --update-state at the same time that it was in use by mysqld.
- Multiple mysqld servers are using the table and one server performed a REPAIR TABLE or CHECK TABLE on the table while it was in use by another server. In this setup, it is safe to use CHECK TABLE, although you might get the warning from other servers. However, REPAIR TABLE should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

  In general, it is a bad idea to share a data directory among multiple servers. See Section 5.7, "Running Multiple MySQL Servers on the Same Machine", for additional discussion.

13.2 The InnoDB Storage Engine

InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. InnoDB row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. InnoDB stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, InnoDB also supports FOREIGN KEY referential-integrity constraints. You can freely mix InnoDB tables with tables from other MySQL storage engines, even within the same statement.

To determine whether your server supports InnoDB use the SHOW ENGINES statement. See Section 12.4.5.10, “SHOW ENGINES Syntax”.

InnoDB has been designed for maximum performance when processing large data volumes. Its CPU efficiency is probably not matched by any other disk-based relational database engine.

The InnoDB storage engine maintains its own buffer pool for caching data and indexes in main memory. InnoDB stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions). This is different from, for example, MyISAM tables where each table is stored using separate files. InnoDB tables can be very large even on operating systems where file size is limited to 2GB.

Starting from MySQL 4.1.5, the improved Windows installer makes InnoDB the MySQL default storage engine on Windows.

InnoDB is used in production at numerous large database sites requiring high performance. The famous Internet news site Slashdot.org runs on InnoDB. Mytrix, Inc. stores more than 1TB of data in InnoDB, and another site handles an average load of 800 inserts/updates per second in InnoDB.

InnoDB is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see http://www.mysql.com/company/legal/licensing/.

Additional Resources

- A forum dedicated to the InnoDB storage engine is available at http://forums.mysql.com/list.php?22.
- Innobase Oy also hosts several forums, available at http://forums.innodb.com.
- InnoDB Hot Backup enables you to back up a running MySQL database, including InnoDB and MyISAM tables, with minimal disruption to operations while producing a consistent snapshot of the database. When InnoDB Hot Backup is copying InnoDB tables, reads and writes to both InnoDB and MyISAM tables can continue. During the copying of MyISAM tables, reads (but not writes) to those tables are permitted. In addition, InnoDB Hot Backup supports creating compressed backup files, and performing backups of subsets of InnoDB tables. In conjunction with MySQL’s binary log, users can perform point-in-time recovery. InnoDB Hot Backup is commercially licensed by Innobase Oy. For a more complete description of InnoDB Hot Backup, see http://www.innodb.com/products/hot-backup/features/ or download the documentation from http://www.innodb.com/doc/hot_backup/manual.html. You can order trial, term, and perpetual licenses from Innobase at http://www.innodb.com/wp/products/hot-backup/order/.

13.2.1 InnoDB Contact Information

Contact information for Innobase Oy, producer of the InnoDB engine:

Web site: http://www.innodb.com/

Email: innodb_sales_ww at oracle.com or use this contact form: http://www.innodb.com/contact-form

Phone:

```
+358-9-6969 3250 (office, Heikki Tuuri)
+358-40-5617367 (mobile, Heikki Tuuri)
+358-40-5939732 (mobile, Satu Sirén)
```

Address:

```
Innobase Oy Inc.
World Trade Center Helsinki
Aleksanterinkatu 17
P.O.Box 800
00101 Helsinki
Finland
```

13.2.2 InnoDB in MySQL 3.23

Beginning with MySQL 4.0, InnoDB is enabled by default, so the following information applies only to MySQL 3.23.

InnoDB tables are included in the MySQL source distribution starting from 3.23.34a and are activated in the MySQL-Max binaries of the 3.23 series. For Windows, the MySQL-Max binaries are included in the standard distribution.

If you have downloaded a binary version of MySQL that includes support for InnoDB, simply follow the instructions of the MySQL manual for installing a binary version of MySQL. If you have MySQL 3.23 installed, the simplest way to install MySQL-Max is to replace the executable mysqld server with the corresponding executable from the MySQL-Max distribution. MySQL and MySQL-Max differ only in the server executable. See Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”, and Section 5.2, “The mysqld-max Extended MySQL Server”.

To compile the MySQL source code with InnoDB support, download MySQL 3.23.34a or newer from http://www.mysql.com/ and configure MySQL with the --with-innodb option. See Section 2.9, “Installing MySQL from Source”.

To use InnoDB tables with MySQL 3.23, you must specify configuration parameters in the [mysqld] section of the my.cnf option file. On Windows, you can use my.ini instead. If you do not configure InnoDB in the option file, InnoDB does not start. (From MySQL 4.0 on, InnoDB uses default parameters if you do not specify any. However, to get best performance, it is still recommended that you use parameters appropriate for your system, as discussed in Section 13.2.3, “InnoDB Configuration”.)

In MySQL 3.23, you must specify at the minimum an innodb_data_file_path [1067] value to configure the InnoDB data files. For example, to configure InnoDB to use a single 500MB data file, place the following setting in the [mysqld] section of your option file:

```
[mysqld]
innodb_data_file_path=ibdata1:500M
```

InnoDB creates the ibdata1 file in the MySQL data directory by default. To specify the location explicitly, specify an innodb_data_home_dir [1068] setting. See Section 13.2.3, “InnoDB Configuration”.

13.2.3 InnoDB Configuration

To enable InnoDB tables in MySQL 3.23, see Section 13.2.2, “InnoDB in MySQL 3.23”.

From MySQL 4.0 on, the InnoDB storage engine is enabled by default. If you do not want to use InnoDB tables, start the server with the --skip-innodb [1066] option to disable the InnoDB storage engine.

In this case, the server will not start if the default storage engine is set to InnoDB. Use --default-storage-engine [386] to set the default to some other engine if necessary.

Caution

InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write operations to improve performance. On some operating systems, the very fsync() system call that should wait until all unwritten data for a file has been flushed might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, you should perform some "pull-the-plug" tests before using anything in production. On Mac OS X 10.3 and up, InnoDB uses a special fcntl() file flush method. Under Linux, it is advisable to **disable the write-back cache**.

On ATA/SATA disk drives, a command such hdparm -W0 /dev/hda may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**

Two important disk-based resources managed by the InnoDB storage engine are its tablespace data files and its log files. If you specify no InnoDB configuration options, MySQL 4.0 and above create an auto-extending 10MB data file named ibdata1 and two 5MB log files named ib_logfile0 and ib_logfile1 in the MySQL data directory. (In MySQL 4.0.0 and 4.0.1, the data file is 64MB and not auto-extending.) In MySQL 3.23, InnoDB does not start if you provide no configuration options. To get good performance, you should explicitly provide InnoDB parameters as discussed in the following examples. Naturally, you should edit the settings to suit your hardware and requirements.

The examples shown here are representative. See Section 13.2.4, "InnoDB Startup Options and System Variables" for additional information about InnoDB-related configuration parameters.

To set up the InnoDB tablespace files, use the innodb_data_file_path [1067] option in the [mysqld] section of the my.cnf option file. On Windows, you can use my.ini instead. The value of innodb_data_file_path [1067] should be a list of one or more data file specifications. If you name more than one data file, separate them by semicolon (";") characters:

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...
```

For example, the following setting explicitly creates a tablespace having the same characteristics as the MySQL 4.0 default:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

This setting configures a single 10MB data file named ibdata1 that is auto-extending. No location for the file is given, so by default, InnoDB creates it in the MySQL data directory.

Sizes are specified using K, M, or G suffix letters to indicate units of KB, MB, or GB.

A tablespace containing a fixed-size 50MB data file named ibdata1 and a 50MB auto-extending file named ibdata2 in the data directory can be configured like this:

```
```

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The full syntax for a data file specification includes the file name, its size, and several optional attributes:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

The `autoextend` and `max` attributes can be used only for the last data file in the `innodb_data_file_path` [1067] line. `autoextend` is available starting from MySQL 3.23.50 and 4.0.2.

If you specify the `autoextend` option for the last data file, `InnoDB` extends the data file if it runs out of free space in the tablespace. The increment is 8MB at a time by default. To modify the increment, change the `innodb_autoextend_increment` [1067] system variable.

If the disk becomes full, you might want to add another data file on another disk. For tablespace reconfiguration instructions, see Section 13.2.6, “Adding, Removing, or Resizing `InnoDB` Data and Log Files”.

`InnoDB` is not aware of the file system maximum file size, so be cautious on file systems where the maximum file size is a small value such as 2GB. To specify a maximum size for an auto-extending data file, use the `max` attribute following the `autoextend` attribute. The following configuration permits `ibdata1` to grow up to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

`InnoDB` creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the `innodb_data_home_dir` [1068] option. For example, to use two files named `ibdata1` and `ibdata2` but create them in the `/ibdata` directory, configure `InnoDB` like this:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

Note

`InnoDB` does not create directories, so make sure that the `/ibdata` directory exists before you start the server. This is also true of any log file directories that you configure. Use the Unix or DOS `mkdir` command to create any necessary directories.

Make sure that the MySQL server has the proper access rights to create files in the data directory. More generally, the server must have access rights in any directory where it needs to create data files or log files.

`InnoDB` forms the directory path for each data file by textually concatenating the value of `innodb_data_home_dir` [1068] to the data file name, adding a path name separator (slash or backslash) between values if necessary. If the `innodb_data_home_dir` [1068] option is not mentioned in `my.cnf` at all, the default value is the “dot” directory `./`, which means the MySQL data directory. (The MySQL server changes its current working directory to its data directory when it begins executing.)

If you specify `innodb_data_home_dir` [1068] as an empty string, you can specify absolute paths for the data files listed in the `innodb_data_file_path` [1067] value. The following example is equivalent to the preceding one:

```
[mysqld]
```

```
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

A simple my.cnf example. Suppose that you have a computer with 512MB RAM and one hard disk. The following example shows possible configuration parameters in my.cnf or my.ini for InnoDB. The example assumes the use of MySQL-Max 3.23.50 or later or MySQL 4.0.2 or later because it uses the autoextend attribute. The example suits most users, both on Unix and Windows, who do not want to distribute InnoDB data files and log files onto several disks. It creates an auto-extending data file ibdata1 and two InnoDB log files ib_logfile0 and ib_logfile1 in the MySQL data directory. Also, the small archived InnoDB log file ib_arch_log_0000000000 that InnoDB creates automatically ends up in the data directory.

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
innodb_buffer_pool_size=256M
innodb_additional_mem_pool_size=20M
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=64M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Note that data files must be less than 2GB in some file systems. The combined size of the log files must be less than 4GB. The combined size of data files must be at least 10MB.

When you create an InnoDB tablespace for the first time, it is best that you start the MySQL server from the command prompt. InnoDB then prints the information about the database creation to the screen, so you can see what is happening. For example, on Windows, if mysqld is located in C:\Program Files\MySQL\MySQL Server 4.1\bin, you can start it like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 4.1\bin\mysqld" --console
```

If you do not send server output to the screen, check the server's error log to see what InnoDB prints during the startup process.

For an example of what the information displayed by InnoDB should look like, see Section 13.2.3.3, “Creating the InnoDB Tablespace”.

You can place InnoDB options in the [mysqld] group of any option file that your server reads when it starts. The locations for option files are described in Section 4.2.3.3, “Using Option Files”.

If you installed MySQL on Windows using the installation and configuration wizards, the option file will be the my.ini file located in your MySQL installation directory. See Section 2.3.4.14, “The Location of the my.ini File”.

If your PC uses a boot loader where the C: drive is not the boot drive, your only option is to use the my.ini file in your Windows directory (typically C:\WINDOWS or C:\WINNT). You can use the SET command at the command prompt in a console window to print the value of WINDIR:

```
C:\> SET WINDIR
windir=C:\WINDOWS
```

To make sure that mysqld reads options only from a specific file, use the --defaults-file [235] option as the first option on the command line when starting the server:

```
mysqld --defaults-file=your_path_to_my_cnf
```

An advanced my.cnf example. Suppose that you have a Linux computer with 2GB RAM and three 60GB hard disks at directory paths /, /dr2 and /dr3. The following example shows possible configuration parameters in my.cnf for InnoDB.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /ibdata/ibdata1:2000M;/dr2/ibdata/ibdata2:2000M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
# innodb_log_arch_dir must be the same as innodb_log_group_home_dir
# (starting from 4.0.6, you can omit it)
innodb_log_arch_dir = /dr3/iblogs
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
# Uncomment the next line if you want to use it
#innodb_thread_concurrency=5
```

In some cases, database performance improves if the data is not all placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. The example illustrates how to do this. It places the two data files on different disks and places the log files on the third disk. InnoDB fills the tablespace beginning with the first data file. You can also use raw disk partitions (raw devices) as InnoDB data files, which may speed up I/O. See Section 13.2.3.2, “Using Raw Devices for the Shared Tablespace”.

Warning

On 32-bit GNU/Linux x86, you must be careful not to set memory usage too high. glibc may permit the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB:

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL binaries provided by Oracle Corporation.) and in the worst case also uses sort_buffer_size + read_buffer_size additional memory.

> In MySQL 4.1, by compiling MySQL yourself, you can use up to 64GB of physical memory in 32-bit Windows. See the description for innodb_buffer_pool_awe_mem_mb [1067] in Section 13.2.4, “InnoDB Startup Options and System Variables”.

Tuning other mysqld server parameters. The following values are typical and suit most users:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
key_buffer_size=value
```

13.2.3.1 Using Per-Table Tablespaces

> **Note**
>
> There is a known bug in versions prior to 4.1.8 that manifests itself if you specify innodb_file_per_table [1068] in my.cnf. If you shut down mysqld, then records may disappear from the secondary indexes of a table. See Bug #7496 for more information and workarounds. This is fixed in 4.1.9, but another bug (Bug #8021) bit the Windows version in 4.1.9, and in the Windows version of 4.1.9 you must put the line innodb_flush_method=unbuffered to your my.cnf or my.ini to get mysqld to work.

Starting from MySQL 4.1.1, you can store each InnoDB table and its indexes in its own file. This feature is called “multiple tablespaces” because in effect each table has its own tablespace.

Using multiple tablespaces can be beneficial to users who want to move specific tables to separate physical disks or who wish to restore backups of single tables quickly without interrupting the use of other InnoDB tables.

If you need to downgrade to 4.0, you must make table dumps and re-create the whole InnoDB tablespace. If you have not created new InnoDB tables under MySQL 4.1.1 or later, and need to downgrade quickly, you can also do a direct downgrade to the MySQL 4.0.18 or later in the 4.0 series. Before doing the direct downgrade to 4.0.x, you have to end all client connections to the mysqld server that is to be downgraded, and let it run the purge and insert buffer merge operations to completion, so that SHOW INNODB STATUS shows the main thread in the state waiting for server activity. Then you can shut down mysqld and start 4.0.18 or later in the 4.0 series.

To enable multiple tablespaces, start the server with the --innodb_file_per_table [1068] option. For example, add a line to the [mysqld] section of my.cnf:

```
[mysqld]
innodb_file_per_table
```

With multiple tablespaces enabled, InnoDB stores each newly created table into its own *tbl_name*.ibd file in the database directory where the table belongs. This is similar to what the MyISAM storage engine does, but MyISAM divides the table into a *tbl_name*.MYD data file and an *tbl_name*.MYI index file. For InnoDB, the data and the indexes are stored together in the .ibd file. The *tbl_name*.frm file is still created as usual.

You cannot freely move .ibd files between database directories as you can with MyISAM table files. This is because the table definition that is stored in the InnoDB shared tablespace includes the database name, and because InnoDB must preserve the consistency of transaction IDs and log sequence numbers.

If you remove the innodb_file_per_table [1068] line from my.cnf and restart the server, InnoDB creates tables inside the shared tablespace files again.

The --innodb_file_per_table [1068] option affects only table creation, not access to existing tables. If you start the server with this option, new tables are created using .ibd files, but you can still access tables that exist in the shared tablespace. If you start the server without this option, new tables are created in the shared tablespace, but you can still access any tables that were created using multiple tablespaces.

Note

InnoDB always needs the shared tablespace because it puts its internal data dictionary and undo logs there. The .ibd files are not sufficient for InnoDB to operate.

To move an .ibd file and the associated table from one database to another, use a RENAME TABLE statement:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an .ibd file, you can restore it to the MySQL installation from which it originated as follows:

1. Issue this ALTER TABLE statement to delete the current .ibd file:

   ```
   ALTER TABLE tbl_name DISCARD TABLESPACE;
   ```

2. Copy the backup .ibd file to the proper database directory.

3. Issue this ALTER TABLE statement to tell InnoDB to use the new .ibd file for the table:

   ```
   ALTER TABLE tbl_name IMPORT TABLESPACE;
   ```

In this context, a “clean” .ibd file backup is one for which the following requirements are satisfied:

- There are no uncommitted modifications by transactions in the .ibd file.
- There are no unmerged insert buffer entries in the .ibd file.
- Purge has removed all delete-marked index records from the .ibd file.
- mysqld has flushed all modified pages of the .ibd file from the buffer pool to the file.

You can make a clean backup .ibd file using the following method:

1. Stop all activity from the mysqld server and commit all transactions.
2. Wait until SHOW INNODB STATUS shows that there are no active transactions in the database, and the main thread status of InnoDB is Waiting for server activity. Then you can make a copy of the .ibd file.

Another method for making a clean copy of an .ibd file is to use the commercial InnoDB Hot Backup tool:

1. Use InnoDB Hot Backup to back up the InnoDB installation.

2. Start a second mysqld server on the backup and let it clean up the .ibd files in the backup.

13.2.3.2 Using Raw Devices for the Shared Tablespace

Starting from MySQL 3.23.41, you can use raw disk partitions as data files in the shared tablespace. By using a raw disk, you can perform nonbuffered I/O on Windows and on some Unix systems without file system overhead. This may improve performance, but you are advised to perform tests with and without raw partitions to verify whether this is actually so on your system.

When you create a new data file, you must put the keyword newraw immediately after the data file size in innodb_data_file_path [1067]. The partition must be at least as large as the size that you specify. Note that 1MB in InnoDB is 1024 × 1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

The next time you start the server, InnoDB notices the newraw keyword and initializes the new partition. However, do not create or change any InnoDB tables yet. Otherwise, when you next restart the server, InnoDB reinitializes the partition and your changes are lost. (Starting from MySQL 3.23.44, as a safety measure InnoDB prevents users from modifying data when any partition with newraw is specified.)

After InnoDB has initialized the new partition, stop the server, change newraw in the data file specification to raw:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Graw;/dev/hdd2:2Graw
```

Then restart the server and InnoDB permits changes to be made.

On Windows, starting from 4.1.1, you can allocate a disk partition as a data file like this:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

The //./ corresponds to the Windows syntax of \\.\ for accessing physical drives.

When you use a raw disk partition, be sure that it has permissions that enable read and write access by the account used for running the MySQL server. For example, if you run the server as the mysql user, the partition must permit read and write access to mysql. If you run the server with the --memlock [389] option, the server must be run as root, so the partition must permit access to root.

13.2.3.3 Creating the InnoDB Tablespace

Suppose that you have installed MySQL and have edited your option file so that it contains the necessary InnoDB configuration parameters. Before starting MySQL, you should verify that the directories you have specified for InnoDB data files and log files exist and that the MySQL server has access rights to those directories. InnoDB does not create directories, only files. Check also that you have enough disk space for the data and log files.

It is best to run the MySQL server mysqld from the command prompt when you first start the server with InnoDB enabled, not from mysqld_safe or as a Windows service. When you run from a command prompt you see what mysqld prints and what is happening. On Unix, just invoke mysqld. On Windows, start mysqld with the --console [385] option to direct the output to the console window.

When you start the MySQL server after initially configuring InnoDB in your option file, InnoDB creates your data files and log files, and prints something like this:

```
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size
to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size
to 5242880
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
InnoDB: Started
mysqld: ready for connections
```

At this point InnoDB has initialized its tablespace and log files. You can connect to the MySQL server with the usual MySQL client programs like mysql. When you shut down the MySQL server with mysqladmin shutdown, the output is like this:

```
010321 18:33:34  mysqld: Normal shutdown
010321 18:33:34  mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

You can look at the data file and log directories and you see the files created there. The log directory also contains a small file named ib_arch_log_0000000000. That file resulted from the database creation, after which InnoDB switched off log archiving. When MySQL is started again, the data files and log files have been created already, so the output is much briefer:

```
InnoDB: Started
mysqld: ready for connections
```

Starting from MySQL 4.1.1, you can add the option innodb_file_per_table [1068] to my.cnf to make InnoDB store each table to its own .ibd file in the same MySQL database directory where the .frm file is created. See Section 13.2.3.1, “Using Per-Table Tablespaces”.

13.2.3.4 Dealing with InnoDB Initialization Problems

If InnoDB prints an operating system error during a file operation, usually the problem has one of the following causes:

- You did not create the InnoDB data file directory or the InnoDB log directory.
- mysqld does not have access rights to create files in those directories.
- mysqld cannot read the proper my.cnf or my.ini option file, and consequently does not see the options that you specified.
- The disk is full or a disk quota is exceeded.

- You have created a subdirectory whose name is equal to a data file that you specified, so the name cannot be used as a file name.
- There is a syntax error in the innodb_data_home_dir [1068] or innodb_data_file_path [1067] value.

If something goes wrong when InnoDB attempts to initialize its tablespace or its log files, you should delete all files created by InnoDB. This means all ibdata files and all ib_logfile files. In case you have already created some InnoDB tables, delete the corresponding .frm files for these tables (and any .ibd files if you are using multiple tablespaces) from the MySQL database directories as well. Then you can try the InnoDB database creation again. It is best to start the MySQL server from a command prompt so that you see what is happening.

13.2.4 InnoDB Startup Options and System Variables

This section describes the InnoDB-related command options and system variables. System variables that take a numeric value can be specified as *--var_name=value* on the command line or as *var_name=value* in option files. Many of the system variables can be changed at runtime (see Section 5.1.4.2, “Dynamic System Variables”). (Before MySQL 4.0.2, system variable values should be specified using --set-variable syntax.) For more information on specifying options and system variables, see Section 4.2.3, “Specifying Program Options”.

Caution

It is not a good idea to configure InnoDB to use data files or log files on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

InnoDB Command Options

- --innodb [1066]

  Enables the InnoDB storage engine, if the server was compiled with InnoDB support.

  To disable InnoDB, use --skip-innodb [1066]. In this case, the server will not start if the default storage engine is set to InnoDB. Use --default-storage-engine [386] to set the default to some other engine if necessary.

- --innodb-status-file [1066]

  Controls whether InnoDB creates a file named innodb_status.*<pid>* in the MySQL data directory. If enabled, InnoDB periodically writes the output of SHOW ENGINE INNODB STATUS to this file.

  By default, the file is not created. To create it, start mysqld with the --innodb-status-file=1 [1066] option. The file is deleted during normal shutdown.

  This option is available as of MySQL 4.0.21.

- --skip-innodb [1066]

  Disable the InnoDB storage engine. See the description of --innodb [1066].

InnoDB System Variables

- innodb_additional_mem_pool_size [1066]

The size in bytes of a memory pool InnoDB uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you need to allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system, and writes warning messages to the MySQL error log. The default value is 1MB.

- innodb_autoextend_increment [1067]

  The increment size (in MB) for extending the size of an auto-extending shared tablespace file when it becomes full. The default value is 8. This variable does not affect the per-table tablespace files that are created if you use innodb_file_per_table=1 [1068]. Those files are auto-extending regardless of the value of innodb_autoextend_increment [1067]. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

  This variable is available starting from MySQL 4.0.24 and 4.1.5. As of MySQL 4.0.24 and 4.1.6, it can be changed at runtime as a global system variable.

- innodb_buffer_pool_awe_mem_mb [1067]

  The size of the buffer pool (in MB), if it is placed in the AWE memory. If it is greater than 0, innodb_buffer_pool_size [1067] is the window in the 32-bit address space of mysqld where InnoDB maps that AWE memory. A good value for innodb_buffer_pool_size [1067] is 500MB. The maximum possible value is 63000.

  To take advantage of AWE memory, you will need to recompile MySQL yourself. The current project settings needed for doing this can be found in the innobase/os/os0proc.c source file.

  This variable is available as of MySQL 4.1.0. It is relevant only in 32-bit Windows. If your 32-bit Windows operating system supports more than 4GB memory, using so-called "Address Windowing Extensions," you can allocate the InnoDB buffer pool into the AWE physical memory using this variable.

- innodb_buffer_pool_size [1067]

  The size in bytes of the memory buffer InnoDB uses to cache data and indexes of its tables. The default value is 8MB. The larger you set this value, the less disk I/O is needed to access data in tables. On a dedicated database server, you may set this to up to 80% of the machine physical memory size. However, do not set it too large because competition for physical memory might cause paging in the operating system. Also, the time to initialize the buffer pool is roughly proportional to its size. On large installations, this initialization time may be significant. For example, on a modern Linux x86_64 server, initialization of a 10GB buffer pool takes approximately 6 seconds. See Section 7.5.2, "The InnoDB Buffer Pool"

- innodb_data_file_path [1067]

  The paths to individual data files and their sizes. The full directory path to each data file is formed by concatenating innodb_data_home_dir [1068] to each path specified here. The file sizes are specified in KB, MB, or GB (1024MB) by appending K, M, or G to the size value. The sum of the sizes of the files must be at least 10MB. If you do not specify On some operating systems, files must be less than 2GB. If you do not specify innodb_data_file_path [1067], the default behavior starting from 4.0 is to create a single 10MB auto-extending data file named ibdata1. Starting from 3.23.44, you can set the file size larger than 4GB on those operating systems that support big files. You can also use raw disk partitions as data files. For detailed information on configuring InnoDB tablespace files, see Section 13.2.3, "InnoDB Configuration".

- innodb_data_home_dir [1068]

  The common part of the directory path for all InnoDB data files in the shared tablespace. This setting does not affect the location of per-file tablespaces when innodb_file_per_table [1068] is enabled. The default value is the MySQL data directory. If you specify the value as an empty string, in which case you can use absolute file paths in innodb_data_file_path [1067].

- innodb_fast_shutdown [1068]

  The InnoDB shutdown mode. The default value is 1 as of MySQL 3.23.50, which causes a “fast” shutdown (the normal type of shutdown). If the value is 0, InnoDB does a full purge and an insert buffer merge before a shutdown. These operations can take minutes, or even hours in extreme cases. If the value is 1, InnoDB skips these operations at shutdown.

- innodb_file_io_threads [1068]

  The number of file I/O threads in InnoDB. Normally, this should be left at the default value of 4, but disk I/O on Windows may benefit from a larger number. On Unix, increasing the number has no effect; InnoDB always uses the default value. This variable is available as of MySQL 3.23.37.

- innodb_file_per_table [1068]

  If innodb_file_per_table [1068] is disabled (the default), InnoDB creates tables in the shared tablespace. If innodb_file_per_table [1068] is enabled, InnoDB creates each new table using its own .ibd file for storing data and indexes, rather than in the shared tablespace. See Section 13.2.3.1, “Using Per-Table Tablespaces”. This variable is available as of MySQL 4.1.1.

  Note

  There is a bug in versions <= 4.1.8 if you specify innodb_file_per_table [1068] in my.cnf! If you shut down mysqld, records may disappear from the secondary indexes of a table. See Bug #7496 for more information and workarounds. This is fixed in 4.1.9, but another bug (Bug #8021) bit the Windows version in 4.1.9, and in the Windows version of 4.1.9, you must put the line innodb_flush_method=unbuffered in your my.cnf or my.ini to get mysqld to work.

- innodb_flush_log_at_trx_commit [1068]

  If the value of innodb_flush_log_at_trx_commit [1068] is 0, the log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit. When the value is 1, the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file. When the value is 2, the log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second also when the value is 2. Note that the once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.

  The default value of this variable is 1 (prior to MySQL 4.0.13, the default is 0).

  A value of 1 is required for ACID compliance. You can achieve better performance by setting the value different from 1, but then you can lose at most one second worth of transactions in a crash. With a value of 0, any mysqld process crash can erase the last second of transactions. With a value of 2, then

only an operating system crash or a power outage can erase the last second of transactions. However, InnoDB's crash recovery is not affected and thus crash recovery does work regardless of the value.

Note

For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, use innodb_flush_log_at_trx_commit=1, sync_binlog=1, and innodb-safe-binlog in your master server my.cnf file.

Caution

Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell mysqld that the flush has taken place, even though it has not. Then the durability of transactions is not guaranteed even with the setting 1, and in the worst case a power outage can even corrupt the InnoDB database. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try using the Unix command hdparm to disable the caching of disk writes in hardware caches, or use some other command specific to the hardware vendor.

- innodb_flush_method [1069]

  If set to fdatasync (the default), InnoDB uses fsync() to flush both the data and log files. If set to O_DSYNC, InnoDB uses O_SYNC to open and flush the log files, and fsync() to flush the data files. If O_DIRECT is specified (available on some GNU/Linux versions starting from MySQL 4.0.14), InnoDB uses O_DIRECT to open the data files, and uses fsync() to flush both the data and log files. Note that starting from MySQL 3.23.41, InnoDB uses fsync() instead of fdatasync(), and it does not use O_DSYNC by default because there have been problems with it on many varieties of Unix. This variable is relevant only for Unix. On Windows, the flush method is always async_unbuffered and cannot be changed. This variable is available as of MySQL 3.23.40.

  Different values of this variable can have a marked effect on InnoDB performance. For example, on some systems where InnoDB data and log files are located on a SAN, it has been found that setting innodb_flush_method [1069] to O_DIRECT can degrade performance of simple SELECT statements by a factor of three.

- innodb_force_recovery [1069]

  The crash recovery mode. Possible values are from 0 to 6. The meanings of these values are described in Section 13.2.7.2, “Forcing InnoDB Recovery”.

Warning

This variable should be set greater than 0 only in an emergency situation when you want to dump your tables from a corrupt database! As a safety measure, InnoDB prevents any changes to its data when this variable is greater than 0. This variable is available starting from MySQL 3.23.44.

- innodb_lock_wait_timeout [1069]

  The timeout in seconds an InnoDB transaction may wait for a lock before being rolled back. The default is 50 seconds.

A lock wait for a MySQL table lock does not happen inside InnoDB, and this timeout does not apply to waits for table locks.

InnoDB does detect transaction deadlocks in its own lock table immediately and rolls back one transaction. The lock wait timeout value does not apply to such a wait.

- innodb_locks_unsafe_for_binlog [1070]

This variable affects how InnoDB uses gap locking for searches and index scans. Normally, InnoDB uses an algorithm called *next-key locking* that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the "gap" before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order. See Section 13.2.9.4, "InnoDB Record, Gap, and Next-Key Locks".

By default, the value of innodb_locks_unsafe_for_binlog [1070] is 0 (disabled), which means that gap locking is enabled: InnoDB uses next-key locks for searches and index scans. To enable the variable, set it to 1. This causes gap locking to be disabled: InnoDB uses only index-record locks for searches and index scans.

Enabling innodb_locks_unsafe_for_binlog [1070] does not disable the use of gap locking for foreign-key constraint checking or duplicate-key checking.

The effect of enabling innodb_locks_unsafe_for_binlog [1070] is similar to but not identical to setting the transaction isolation level to READ COMMITTED [975]:

- Enabling innodb_locks_unsafe_for_binlog [1070] is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.
- innodb_locks_unsafe_for_binlog [1070] can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

READ COMMITTED [975] therefore offers finer and more flexible control than innodb_locks_unsafe_for_binlog [1070]. For additional details about the effect of isolation level on gap locking, see Section 12.3.6, "SET TRANSACTION Syntax".

Enabling innodb_locks_unsafe_for_binlog [1070] may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled. Suppose that there is an index on the id column of the child table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where id is greater than 100. If the locks set on the index records in that range do not lock out inserts made in the gaps, another session can insert a new row into the table. Consequently, if you were to execute the same SELECT again within the same transaction, you would see a new row in the result set returned by the query. This also means that if new items are added to the database, InnoDB does not guarantee serializability. Therefore, if innodb_locks_unsafe_for_binlog [1070] is enabled InnoDB guarantees at most an isolation

level of READ COMMITTED [975]. (Conflict serializability is still guaranteed.) For additional information about phantoms, see Section 13.2.9.5, “Avoiding the Phantom Problem Using Next-Key Locking”.

innodb_locks_unsafe_for_binlog [1070] is available as of MySQL 4.1.4.

- innodb_log_arch_dir [1071]

  The directory where fully written log files would be archived if we used log archiving. The value of this variable should currently be set the same as innodb_log_group_home_dir [1071]. Starting from MySQL 4.0.6, there is no need to set this variable.

- innodb_log_archive [1071]

  Whether to log InnoDB archive files. This variable is unused. Recovery from a backup is done by MySQL using its own log files, so there is no need to archive InnoDB log files. The default for this variable is 0.

- innodb_log_buffer_size [1071]

  The size in bytes of the buffer that InnoDB uses to write to the log files on disk. The default value is 1MB. Sensible values range from 1MB to 8MB. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.

- innodb_log_file_size [1071]

  The size in bytes of each log file in a log group. The combined size of log files must be less than 4GB. The default value is 5MB. Sensible values range from 1MB to 1/*N*-th of the size of the buffer pool, where *N* is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery is slower in case of a crash.

- innodb_log_files_in_group [1071]

  The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) value is 2.

- innodb_log_group_home_dir [1071]

  The directory path to the InnoDB log files. It must have the same value as innodb_log_arch_dir [1071]. If you do not specify any InnoDB log variables, the default is to create two 5MB files names ib_logfile0 and ib_logfile1 in the MySQL data directory.

- innodb_max_dirty_pages_pct [1071]

  This is an integer in the range from 0 to 100. The default value is 90. The main thread in InnoDB tries to write pages from the buffer pool so that the percentage of dirty (not yet written) pages will not exceed this value. Available starting from 4.0.13 and 4.1.1.

- innodb_max_purge_lag [1071]

  This variable controls how to delay INSERT, UPDATE, and DELETE operations when the purge operations are lagging (see Section 13.2.10, “InnoDB Multi-Versioning”). The default value of this

variable is 0, meaning that there are no delays. innodb_max_purge_lag [1071] is available as of MySQL 4.0.22 and 4.1.6.

The InnoDB transaction system maintains a list of transactions that have index records delete-marked by UPDATE or DELETE operations. Let the length of this list be *purge_lag*. When *purge_lag* exceeds innodb_max_purge_lag [1071], each INSERT, UPDATE, and DELETE operation is delayed by ((*purge_lag*/innodb_max_purge_lag [1071])×10)–5 milliseconds. The delay is computed in the beginning of a purge batch, every ten seconds. The operations are not delayed if purge cannot run because of an old consistent read view that could see the rows to be purged.

A typical setting for a problematic workload might be 1 million, assuming that transactions are small, only 100 bytes in size, and it is permissible to have 100MB of unpurged InnoDB table rows.

The lag value is displayed as the history list length in the TRANSACTIONS section of InnoDB Monitor output. For example, if the output includes the following lines, the lag value is 20:

```
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

- innodb_mirrored_log_groups [1072]

  The number of identical copies of log groups to keep for the database. This should be set to 1.

- innodb_open_files [1072]

  This variable is relevant only if you use multiple tablespaces in InnoDB. It specifies the maximum number of .ibd files that InnoDB can keep open at one time. The minimum value is 10. The default value is 300. This variable is available as of MySQL 4.1.1.

  The file descriptors used for .ibd files are for InnoDB only. They are independent of those specified by the --open-files-limit [391] server option, and do not affect the operation of the table cache.

- innodb-safe-binlog

  If this option is given, then after a crash recovery by InnoDB, mysqld truncates the binary log after the last not-rolled-back transaction in the log. The option also causes InnoDB to print an error if the binary log is smaller or shorter than it should be. See Section 5.3.4, “The Binary Log”.

- innodb_table_locks [1072]

  Starting from MySQL 4.0.20, and 4.1.2, InnoDB honors LOCK TABLES. If autocommit = 0 [406], InnoDB honors LOCK TABLES; MySQL does not return from LOCK TABLES ... WRITE until all other threads have released all their locks to the table. The default value of innodb_table_locks [1072] is 1, which means that LOCK TABLES causes InnoDB to lock a table internally if autocommit = 0 [406].

- innodb_thread_concurrency [1072]

  InnoDB tries to keep the number of operating system threads concurrently inside InnoDB less than or equal to the limit given by this variable. The default value is 8. If you have low performance and SHOW INNODB STATUS reveals many threads waiting for semaphores, you may have thread thrashing and

should try setting this variable lower or higher. If you have a computer with many processors and disks, you can try setting the value higher to better utilize the resources of your computer. A recommended value is 2 times the number of CPUs plus the number of disks. A value of 500 or greater disables the concurrency checking. This variable is available starting from MySQL 3.23.44 and 4.0.1.

- sync_binlog [1184]

  If the value of this variable is greater than 0, the MySQL server synchronizes its binary log to disk (using fdatasync()) after every sync_binlog [1184] writes to the binary log. There is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value of sync_binlog [1184] is 0, which does no synchronizing to disk. A value of 1 is the safest choice, because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast). This variable was added in MySQL 4.1.3.

13.2.5 Creating and Using InnoDB Tables

To create an InnoDB table, specify an ENGINE = InnoDB option in the CREATE TABLE statement:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.

The statement creates a table and an index on column a in the InnoDB tablespace that consists of the data files that you specified in my.cnf. In addition, MySQL creates a file customers.frm in the test directory under the MySQL database directory. Internally, InnoDB adds an entry for the table to its own data dictionary. The entry includes the database name. For example, if test is the database in which the customers table is created, the entry is for 'test/customers'. This means you can create a table of the same name customers in some other database, and the table names do not collide inside InnoDB.

You can query the amount of free space in the InnoDB tablespace by issuing a SHOW TABLE STATUS statement for any InnoDB table. The amount of free space in the tablespace appears in the Comment section in the output of SHOW TABLE STATUS. For example:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

The statistics SHOW displays for InnoDB tables are only approximate. They are used in SQL optimization. Table and index reserved sizes in bytes are accurate, though.

13.2.5.1 How to Use Transactions in InnoDB with Different APIs

By default, each client that connects to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. To use multiple-statement transactions, you can switch autocommit off with the SQL statement SET autocommit = 0 and end each transaction with either COMMIT and ROLLBACK. If you want to leave autocommit on, you can begin your transactions within START TRANSACTION and end them with COMMIT or ROLLBACK. Before MySQL 4.0.11, you have to use the keyword BEGIN instead of START TRANSACTION. The following example shows two transactions. The first is committed and the second is rolled back.

```
shell> mysql test

mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (A))
    -> TYPE=InnoDB;
```

```
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+------+--------+
| a    | b      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as `COMMIT` to the MySQL server as strings just like any other SQL statements such as `SELECT` or `INSERT`. Some APIs also offer separate special transaction commit and rollback functions or methods.

13.2.5.2 Converting Tables from Other Storage Engines to InnoDB

To convert a non-`InnoDB` table to use `InnoDB` use `ALTER TABLE`:

```
ALTER TABLE t1 TYPE=InnoDB;
```

Important

Do not convert MySQL system tables in the `mysql` database (such as `user` or `host`) to the `InnoDB` type. This is an unsupported operation. The system tables must always be of the `MyISAM` type.

`InnoDB` does not have a special optimization for separate index creation the way the `MyISAM` storage engine does. Therefore, it does not pay to export and import the table and create indexes afterward. The fastest way to alter a table to `InnoDB` is to do the inserts directly to an `InnoDB` table. That is, use `ALTER TABLE ... TYPE=INNODB`, or create an empty `InnoDB` table with identical definitions and insert the rows with `INSERT INTO ... SELECT * FROM ...`.

If you have `UNIQUE` constraints on secondary keys, starting from MySQL 3.23.52, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET unique_checks=0;
... import operation ...
SET unique_checks=1;
```

For big tables, this saves a lot of disk I/O because `InnoDB` can then use its insert buffer to write secondary index records as a batch. Be certain that the data contains no duplicate keys. `unique_checks` [433] permits but does not require storage engines to ignore duplicate keys.

To get better control over the insertion process, it might be good to insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
   WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records have been inserted, you can rename the tables.

During the conversion of big tables, you should increase the size of the InnoDB buffer pool to reduce disk I/O. Do not use more than 80% of the physical memory, though. You can also increase the sizes of the InnoDB log files.

Make sure that you do not fill up the tablespace: InnoDB tables require a lot more disk space than MyISAM tables. If an ALTER TABLE operation runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see Section 13.2.7.2, “Forcing InnoDB Recovery”.

If you want all your (nonsystem) tables to be created as InnoDB tables, you can, starting from the MySQL 3.23.43, add the line `default-table-type=innodb` to the `[mysqld]` section of your server option file.

13.2.5.3 AUTO_INCREMENT Handling in InnoDB

If you specify an AUTO_INCREMENT column for an InnoDB table, the table handle in the InnoDB data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. This counter is stored only in main memory, not on disk.

InnoDB uses the following algorithm to initialize the auto-increment counter for a table `t` that contains an AUTO_INCREMENT column named `ai_col`: After a server startup, for the first insert into a table `t`, InnoDB executes the equivalent of this statement:

```
SELECT MAX(ai_col) FROM t FOR UPDATE;
```

InnoDB increments by one the value retrieved by the statement and assigns it to the column and to the auto-increment counter for the table. If the table is empty, InnoDB uses the value `1`. If a user invokes a SHOW TABLE STATUS statement that displays output for the table `t` and the auto-increment counter has not been initialized, InnoDB initializes but does not increment the value and stores it for use by later inserts. This initialization uses a normal exclusive-locking read on the table and the lock lasts to the end of the transaction.

InnoDB follows the same procedure for initializing the auto-increment counter for a freshly created table.

After the auto-increment counter has been initialized, if a user does not explicitly specify a value for an AUTO_INCREMENT column, InnoDB increments the counter by one and assigns the new value to the column. If the user inserts a row that explicitly specifies the column value, and the value is bigger than the current counter value, the counter is set to the specified column value.

When accessing the auto-increment counter, InnoDB uses a special table-level AUTO-INC lock that it keeps to the end of the current SQL statement, not to the end of the transaction. The special lock release strategy was introduced to improve concurrency for inserts into a table containing an AUTO_INCREMENT column. Nevertheless, two transactions cannot have the AUTO-INC lock on the same table simultaneously, which can have a performance impact if the AUTO-INC lock is held for a long time. That might be the case for a statement such as INSERT INTO t1 ... SELECT ... FROM t2 that inserts all rows from one table into another.

InnoDB uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, InnoDB reinitializes the counter for each table for the first INSERT to the table, as described earlier.

You may see gaps in the sequence of values assigned to the AUTO_INCREMENT column if you roll back transactions that have generated numbers using the counter.

If a user specifies NULL or 0 for the AUTO_INCREMENT column in an INSERT, InnoDB treats the row as if the value had not been specified and generates a new value for it.

The behavior of the auto-increment mechanism is not defined if a user assigns a negative value to the column or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

An AUTO_INCREMENT column must appear as the first column in an index on an InnoDB table.

Beginning with MySQL 4.1.12, InnoDB supports the AUTO_INCREMENT = *N* table option in ALTER TABLE statements, to set the initial counter value or alter the current counter value. The same is true as of MySQL 4.1.14 for CREATE TABLE. The effect of this option is canceled by a server restart, for reasons discussed earlier in this section.

13.2.5.4 FOREIGN KEY Constraints

Starting from MySQL 3.23.44, InnoDB features foreign key constraints.

InnoDB supports foreign key constraints. The syntax for a foreign key constraint definition in InnoDB looks like this:

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

index_name represents a foreign key ID. If given, this is ignored if an index for the foreign key is defined explicitly. Otherwise, if InnoDB creates an index for the foreign key, it uses *index_name* for the index name.

Foreign keys definitions are subject to the following conditions:

- Both tables must be InnoDB tables and they must not be TEMPORARY tables.
- Corresponding columns in the foreign key and the referenced key must have similar internal data types inside InnoDB so that they can be compared without a type conversion. *The size and sign of integer types must be the same*. The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.
- InnoDB requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist. (This is in contrast to versions older than MySQL 4.1.2, in which indexes had to be created explicitly or the creation of foreign key constraints would fail.) *index_name*, if given, is used as described previously.

- InnoDB permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.
- Index prefixes on foreign key columns are not supported. One consequence of this is that BLOB and TEXT columns cannot be included in a foreign key because indexes on those columns must always include a prefix length.
- If the CONSTRAINT *symbol* clause is given, the *symbol* value must be unique in the database. If the clause is not given, InnoDB creates the name automatically.

InnoDB rejects any INSERT or UPDATE operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table. The action InnoDB takes for any UPDATE or DELETE operation that attempts to update or delete a candidate key value in the parent table that has some matching rows in the child table is dependent on the *referential action* specified using ON UPDATE and ON DELETE subclauses of the FOREIGN KEY clause. When the user attempts to delete or update a row from a parent table, and there are one or more matching rows in the child table, InnoDB supports five options regarding the action to be taken. If ON DELETE or ON UPDATE are not specified, the default action is RESTRICT.

- CASCADE: Delete or update the row from the parent table and automatically delete or update the matching rows in the child table. ON DELETE CASCADE is supported starting from MySQL 3.23.50 and ON UPDATE CASCADE is supported starting from 4.0.8. Between two tables, you should not define several ON UPDATE CASCADE clauses that act on the same column in the parent table or in the child table.
- SET NULL: Delete or update the row from the parent table and set the foreign key column or columns in the child table to NULL. This is valid only if the foreign key columns do not have the NOT NULL qualifier specified. ON DELETE SET NULL is available starting from MySQL 3.23.50 and ON UPDATE SET NULL is available starting from 4.0.8.

  If you specify a SET NULL action, *make sure that you have not declared the columns in the child table as NOT NULL*.
- NO ACTION: In standard SQL, NO ACTION means *no action* in the sense that an attempt to delete or update a primary key value will not be permitted to proceed if there is a related foreign key value in the referenced table. Starting from 4.0.18 InnoDB rejects the delete or update operation for the parent table.
- RESTRICT: Rejects the delete or update operation for the parent table. Specifying RESTRICT (or NO ACTION) is the same as omitting the ON DELETE or ON UPDATE clause. (Some database systems have deferred checks, and NO ACTION is a deferred check. In MySQL, foreign key constraints are checked immediately, so NO ACTION is the same as RESTRICT.)
- SET DEFAULT: This action is recognized by the parser, but InnoDB rejects table definitions containing ON DELETE SET DEFAULT or ON UPDATE SET DEFAULT clauses.

InnoDB supports foreign key references within a table. In these cases, “child table records” really refers to dependent records within the same table.

Here is a simple example that relates parent and child tables through a single-column foreign key:

```
CREATE TABLE parent (id INT NOT NULL,
                     PRIMARY KEY (id)
) TYPE=INNODB;
CREATE TABLE child (id INT, parent_id INT,
                    INDEX par_ind (parent_id),
                    FOREIGN KEY (parent_id) REFERENCES parent(id)
```

```
                      ON DELETE CASCADE
) TYPE=INNODB;
```

A more complex example in which a `product_order` table has foreign keys for two other tables. One foreign key references a two-column index in the `product` table. The other references a single-column index in the `customer` table:

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                      price DECIMAL,
                      PRIMARY KEY(category, id)) TYPE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                       PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                            product_category INT NOT NULL,
                            product_id INT NOT NULL,
                            customer_id INT NOT NULL,
                            PRIMARY KEY(no),
                            INDEX (product_category, product_id),
                            FOREIGN KEY (product_category, product_id)
                              REFERENCES product(category, id)
                              ON UPDATE CASCADE ON DELETE RESTRICT,
                            INDEX (customer_id),
                            FOREIGN KEY (customer_id)
                              REFERENCES customer(id)) TYPE=INNODB;
```

Starting from MySQL 3.23.50, `InnoDB` enables you to add a new foreign key constraint to a table by using `ALTER TABLE`:

```
ALTER TABLE tbl_name
    ADD [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

The foreign key can be self referential (referring to the same table). When you add a foreign key constraint to a table using `ALTER TABLE`, *remember to create the required indexes first.*

Starting from MySQL 4.0.13, `InnoDB` supports the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

If the `FOREIGN KEY` clause included a `CONSTRAINT` name when you created the foreign key, you can refer to that name to drop the foreign key. (A constraint name can be given as of MySQL 4.0.18.) Otherwise, the *`fk_symbol`* value is internally generated by `InnoDB` when the foreign key is created. To find out the symbol when you want to drop a foreign key, use the `SHOW CREATE TABLE` statement. For example:

```
mysql> SHOW CREATE TABLE ibtest11c\G
*************************** 1. row ***************************
       Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY  (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
```

```
REFERENCES `ibtest11a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `ibtest11a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) TYPE=InnoDB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

You cannot add a foreign key and drop a foreign key in separate clauses of a single ALTER TABLE statement. Separate statements are required.

If ALTER TABLE for an InnoDB table results in changes to column values (for example, because a column is truncated), InnoDB's FOREIGN KEY constraint checks do not notice possible violations caused by changing the values.

Starting from MySQL 3.23.50, the InnoDB parser permits table and column identifiers in a FOREIGN KEY ... REFERENCES ... clause to be quoted within backticks. (Alternatively, double quotation marks can be used if the ANSI_QUOTES [458] SQL mode is enabled.) The InnoDB parser also takes into account the setting of the lower_case_table_names [418] system variable.

Before MySQL 3.23.50, ALTER TABLE or CREATE INDEX should not be used in connection with tables that have foreign key constraints or that are referenced in foreign key constraints: Any ALTER TABLE removes all foreign key constraints defined for the table. You should not use ALTER TABLE with the referenced table, either. Instead, use DROP TABLE and CREATE TABLE to modify the schema. When MySQL does an ALTER TABLE it may internally use RENAME TABLE, and that confuses the foreign key constraints that refer to the table. In MySQL, a CREATE INDEX statement is processed as an ALTER TABLE, so the same considerations apply.

Starting from MySQL 3.23.50, InnoDB returns the foreign key definitions of a table as part of the output of the SHOW CREATE TABLE statement:

```
SHOW CREATE TABLE tbl_name;
```

mysqldump also produces correct definitions of tables in the dump file, and does not forget about the foreign keys.

You can also display the foreign key constraints for a table like this:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

The foreign key constraints are listed in the Comment column of the output.

When performing foreign key checks, InnoDB sets shared row-level locks on child or parent records it has to look at. InnoDB checks foreign key constraints immediately; the check is not deferred to transaction commit.

To make it easier to reload dump files for tables that have foreign key relationships, mysqldump automatically includes a statement in the dump output to set foreign_key_checks [411] to 0 as of MySQL 4.1.1. This avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. For earlier versions, you can disable the variable manually within mysql when loading the dump file like this:

```
mysql> SET foreign_key_checks = 0;
mysql> SOURCE dump_file_name;
mysql> SET foreign_key_checks = 1;
```

This enables you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. Setting foreign_key_checks [411] to 0 can also be useful for ignoring foreign key constraints during LOAD DATA and ALTER TABLE operations. However, even if foreign_key_checks = 0 [411], InnoDB does not permit the creation of a foreign key constraint where a column references a nonmatching column type. Also, if an InnoDB table has foreign key constraints, ALTER TABLE cannot be used to change the table to use another storage engine. To alter the storage engine, you must drop any foreign key constraints first. foreign_key_checks [411] is available starting from MySQL 3.23.52 and 4.0.3.

InnoDB does not permit you to drop a table that is referenced by a FOREIGN KEY constraint, unless you do SET foreign_key_checks = 0. When you drop a table, the constraints that were defined in its create statement are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the right column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns error number 1005 and refers to error 150 in the error message.

If MySQL reports an error number 1005 from a CREATE TABLE statement, and the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. Similarly, if an ALTER TABLE fails and it refers to error 150, that means a foreign key definition would be incorrectly formed for the altered table. Starting from MySQL 4.0.13, you can use SHOW INNODB STATUS to display a detailed explanation of the latest InnoDB foreign key error in the server.

Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including InnoDB, recognizes or enforces the MATCH clause used in referential-integrity constraint definitions. Use of an explicit MATCH clause will not have the specified effect, and also causes ON DELETE and ON UPDATE clauses to be ignored. For these reasons, specifying MATCH should be avoided.

The MATCH clause in the SQL standard controls how NULL values in a composite (multiple-column) foreign key are handled when comparing to a primary key. Starting from MySQL 3.23.50, InnoDB does not check foreign key constraints on those foreign key or referenced key values that contain a NULL column. InnoDB essentially implements the semantics defined by MATCH SIMPLE, which permit a foreign key to be all or partially NULL. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table.

Additionally, MySQL and InnoDB require that the referenced columns be indexed for performance. However, the system does not enforce a requirement that the referenced columns be UNIQUE or be declared NOT NULL. The handling of foreign key references to nonunique keys or keys that contain NULL values is not well defined for operations such as UPDATE or DELETE CASCADE. You are advised to use foreign keys that reference only UNIQUE and NOT NULL keys.

Furthermore, InnoDB does not recognize or support “inline REFERENCES specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. InnoDB accepts REFERENCES clauses only when specified as part of a separate FOREIGN KEY specification. For other storage engines, MySQL Server parses and ignores foreign key specifications.

Deviation from SQL standards: If there are several rows in the parent table that have the same referenced key value, InnoDB acts in foreign key checks as if the other parent rows with the same key

value do not exist. For example, if you have defined a RESTRICT type constraint, and there is a child row with several parent rows, InnoDB does not permit the deletion of any of those parent rows.

InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

Deviation from SQL standards: A FOREIGN KEY constraint that references a non-UNIQUE key is not standard SQL. It is an InnoDB extension to standard SQL.

Deviation from SQL standards: If ON UPDATE CASCADE or ON UPDATE SET NULL recurses to update the *same table* it has previously updated during the cascade, it acts like RESTRICT. This means that you cannot use self-referential ON UPDATE CASCADE or ON UPDATE SET NULL operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential ON DELETE SET NULL, on the other hand, is possible from 4.0.13. A self-referential ON DELETE CASCADE has been possible since ON DELETE was implemented. Since 4.0.21, cascading operations may not be nested more than 15 levels.

Deviation from SQL standards: Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, InnoDB checks UNIQUE and FOREIGN KEY constraints row-by-row. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the **whole SQL statement** has been processed. Until InnoDB implements deferred constraint checking, some things will be impossible, such as deleting a record that refers to itself using a foreign key.

13.2.5.5 InnoDB and MySQL Replication

MySQL replication works for InnoDB tables as it does for MyISAM tables. It is also possible to use replication in a way where the storage engine on the slave is not the same as the original storage engine on the master. For example, you can replicate modifications to an InnoDB table on the master to a MyISAM table on the slave.

To set up a new slave for a master, you have to make a copy of the InnoDB tablespace and the log files, as well as the .frm files of the InnoDB tables, and move the copies to the slave. If the innodb_file_per_table [1068] variable is enabled, you must also copy the .ibd files as well. For the proper procedure to do this, see Section 13.2.7, “Backing Up and Recovering an InnoDB Database”.

If you can shut down the master or an existing slave, you can take a cold backup of the InnoDB tablespace and log files and use that to set up a slave. To make a new slave without taking down any server you can also use the commercial InnoDB Hot Backup tool.

There are minor limitations in InnoDB replication:

- LOAD TABLE FROM MASTER does not work for InnoDB type tables. There are workarounds: 1) dump the table on the master and import the dump file into the slave, or 2) use ALTER TABLE *tbl_name* TYPE=MyISAM on the master before setting up replication with LOAD TABLE *tbl_name* FROM MASTER, and then use ALTER TABLE to alter the master table back to the InnoDB type afterward. However, this should not be done for tables that have foreign key definitions because the definitions will be lost.

- Before MySQL 4.0.6, SLAVE STOP did not respect the boundary of a multiple-statement transaction. An incomplete transaction would be rolled back, and the next SLAVE START would only execute the remaining part of the half transaction. That would cause replication to fail.

- Before MySQL 4.0.6, a slave crash in the middle of a multiple-statement transaction would cause the same problem as SLAVE STOP.

- Before MySQL 4.0.11, replication of the SET foreign_key_checks = 0 statement does not work properly.

Most of these limitations can be eliminated by using more recent server versions for which the limitations do not apply.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to slaves. See Section 12.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

Replication and CASCADE. Cascading actions for InnoDB tables on the master are replicated on the slave *only* if the tables sharing the foreign key relation use InnoDB on both the master and slave. Suppose that you have started replication, and then create two tables on the master using the following CREATE TABLE statements:

```
CREATE TABLE fc1 (
    i INT PRIMARY KEY,
    j INT
) ENGINE = InnoDB;

CREATE TABLE fc2 (
    m INT PRIMARY KEY,
    n INT,
    FOREIGN KEY ni (n) REFERENCES fc1 (i)
        ON DELETE CASCADE
) ENGINE = InnoDB;
```

Suppose that the slave does not have InnoDB support enabled. If this is the case, then the tables on the slave are created, but they use the MyISAM storage engine, and the FOREIGN KEY option is ignored. Now we insert some rows into the tables on the master:

```
master> INSERT INTO fc1 VALUES (1, 1), (2, 2);
Query OK, 2 rows affected (0.09 sec)
Records: 2  Duplicates: 0  Warnings: 0

master> INSERT INTO fc2 VALUES (1, 1), (2, 2), (3, 1);
Query OK, 3 rows affected (0.19 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

At this point, on both the master and the slave, table fc1 contains 2 rows, and table fc2 contains 3 rows, as shown here:

```
master> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

master> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)

slave> SELECT * FROM fc1;
```

```
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

slave> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)
```

Now suppose that you perform the following DELETE statement on the master:

```
master> DELETE FROM fc1 WHERE i=1;
Query OK, 1 row affected (0.09 sec)
```

Due to the cascade, table fc2 on the master now contains only 1 row:

```
master> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 2 | 2 |
+---+---+
1 row in set (0.00 sec)
```

However, the cascade does not propagate on the slave because on the slave the DELETE for fc1 deletes no rows from fc2. The slave's copy of fc2 still contains all of the rows that were originally inserted:

```
slave> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 3 | 1 |
| 2 | 2 |
+---+---+
3 rows in set (0.00 sec)
```

This difference is due to the fact that the cascading deletes are handled internally by the InnoDB storage engine, which means that none of the changes are logged.

13.2.6 Adding, Removing, or Resizing InnoDB Data and Log Files

This section describes what you can do when your InnoDB tablespace runs out of room or when you want to change the size of the log files.

From MySQL 3.23.50 and 4.0.2, the easiest way to increase the size of the InnoDB tablespace is to configure it from the beginning to be auto-extending. Specify the autoextend attribute for the last data file in the tablespace definition. Then InnoDB increases the size of that file automatically in 8MB increments when it runs out of space. Starting with MySQL 4.0.24 and 4.1.5, the increment size can be changed by setting the value of the innodb_autoextend_increment [1067] system variable, which is measured in MB.

Alternatively, you can increase the size of your tablespace by adding another data file. To do this, you have to shut down the MySQL server, change the tablespace configuration to add a new data file to the end of innodb_data_file_path [1067], and start the server again.

If your last data file was defined with the keyword autoextend, the procedure for reconfiguring the tablespace must take into account the size to which the last data file has grown. Obtain the size of the data file, round it down to the closest multiple of 1024 × 1024 bytes (= 1MB), and specify the rounded size explicitly in innodb_data_file_path [1067]. Then you can add another data file. Remember that only the last data file in the innodb_data_file_path [1067] can be specified as auto-extending.

As an example, assume that the tablespace has just one auto-extending data file ibdata1:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that this data file, over time, has grown to 988MB. Here is the configuration line after modifying the original data file to not be auto-extending and adding another auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When you add a new file to the tablespace configuration, make sure that it does not exist. InnoDB will create and initialize the file when you restart the server.

Currently, you cannot remove a data file from the tablespace. To decrease the size of your tablespace, use this procedure:

1. Use mysqldump to dump all your InnoDB tables.
2. Stop the server.
3. Remove all the existing tablespace files, including the ibdata and ib_log files. If you want to keep a backup copy of the information, then copy all the ib* files to another location before the removing the files in your MySQL installation.
4. Remove any .frm files for InnoDB tables.
5. Configure a new tablespace.
6. Restart the server.
7. Import the dump files.

If you want to change the number or the size of your InnoDB log files, stop the MySQL server and make sure that it shuts down without errors (to ensure that there is no information for outstanding transactions in the log). Copy the old log files into a safe place in case something went wrong during the shutdown and you need them to recover the tablespace. Delete the old log files from the log file directory, edit my.cnf to change the log file configuration, and start the MySQL server again. mysqld sees that no InnoDB log files exist at startup and creates new ones.

13.2.7 Backing Up and Recovering an InnoDB Database

The key to safe database management is making regular backups.

InnoDB Hot Backup enables you to back up a running MySQL database, including InnoDB and MyISAM tables, with minimal disruption to operations while producing a consistent snapshot of the database. When InnoDB Hot Backup is copying InnoDB tables, reads and writes to both InnoDB

and MyISAM tables can continue. During the copying of MyISAM tables, reads (but not writes) to those tables are permitted. In addition, InnoDB Hot Backup supports creating compressed backup files, and performing backups of subsets of InnoDB tables. In conjunction with MySQL's binary log, users can perform point-in-time recovery. InnoDB Hot Backup is commercially licensed by Innobase Oy. For a more complete description of InnoDB Hot Backup, see http://www.innodb.com/products/hot-backup/features/ or download the documentation from http://www.innodb.com/doc/hot_backup/manual.html. You can order trial, term, and perpetual licenses from Innobase at http://www.innodb.com/wp/products/hot-backup/order/.

If you are able to shut down your MySQL server, you can make a binary backup that consists of all files used by InnoDB to manage its tables. Use the following procedure:

1. Shut down the MySQL server and make sure that it stops without errors.
2. Copy all InnoDB data files (ibdata files and .ibd files) into a safe place.
3. Copy all the .frm files for InnoDB tables to a safe place.
4. Copy all InnoDB log files (ib_logfile files) to a safe place.
5. Copy your my.cnf configuration file or files to a safe place.

In addition to making binary backups as just described, you should also regularly make dumps of your tables with mysqldump. The reason for this is that a binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. mysqldump also has a --single-transaction [298] option for making a consistent snapshot without locking out other clients. See Section 6.3.1, "Establishing a Backup Policy".

Replication works with InnoDB tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability.

To be able to recover your InnoDB database to the present from the time at which the binary backup was made, you must run your MySQL server with binary logging turned on. To achieve point-in-time recovery after restoring a backup, you can apply changes from the binary log that occurred after the backup was made. See Section 6.5, "Point-in-Time (Incremental) Recovery Using the Binary Log".

To recover from a crash of your MySQL server, the only requirement is to restart it. InnoDB automatically checks the logs and performs a roll-forward of the database to the present. InnoDB automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, mysqld displays output something like this:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
```

```
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

If your database becomes corrupted or disk failure occurs, you must perform the recovery using a backup. In the case of corruption, you should first find a backup that is not corrupted. After restoring the base backup, do a point-in-time recovery from the binary log files using `mysqlbinlog` and `mysql` to restore the changes that occurred after the backup was made.

In some cases of database corruption it is enough just to dump, drop, and re-create one or a few corrupt tables. You can use the `CHECK TABLE` SQL statement to check whether a table is corrupt, although `CHECK TABLE` naturally cannot detect every possible kind of corruption. You can use the Tablespace Monitor to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best first to try restarting your computer. Doing so may eliminate errors that appeared to be database page corruption.

13.2.7.1 The InnoDB Recovery Process

`InnoDB` crash recovery consists of several steps. The first step, redo log application, is performed during the initialization, before accepting any connections. If all changes were flushed from the buffer pool to the tablespaces (`ibdata*` and `*.ibd` files) at the time of the shutdown or crash, the redo log application can be skipped. If the redo log files are missing at startup, `InnoDB` skips the redo log application.

The remaining steps after redo log application do not depend on the redo log (other than for logging the writes) and are performed in parallel with normal processing. These include:

- Rolling back incomplete transactions: Any transactions that were active at the time of crash or fast shutdown.
- Insert buffer merge: Applying changes from the insert buffer tree (from the shared tablespace) to leaf pages of secondary indexes as the index pages are read to the buffer pool.
- Purge: Deleting delete-marked records that are no longer visible for any active transaction.

Of these, only rollback of incomplete transactions is special to crash recovery. The insert buffer merge and the purge are performed during normal processing.

13.2.7.2 Forcing InnoDB Recovery

If there is database page corruption, you may want to dump your tables from the database with `SELECT INTO ... OUTFILE`. Usually, most of the data obtained in this way is intact. However, it is possible that the corruption might cause `SELECT * FROM tbl_name` statements or `InnoDB` background operations to crash or assert, or even cause `InnoDB` roll-forward recovery to crash. In such cases, starting from MySQL 3.23.44, you can use the `innodb_force_recovery` [1069] option to force the `InnoDB` storage engine to start up, and you can also prevent background operations from running, so that you are able to dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 4
```

Before MySQL 4.0, use this syntax instead:

```
[mysqld]
set-variable = innodb_force_recovery=4
```

innodb_force_recovery [1069] is 0 by default (normal startup without forced recovery) The permissible nonzero values for innodb_force_recovery [1069] follow. A larger number includes all precautions of smaller numbers. If you are able to dump your tables with an option value of at most 4, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 6 is more drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

- 1 (SRV_FORCE_IGNORE_CORRUPT)

  Let the server run even if it detects a corrupt page. Try to make SELECT * FROM *tbl_name* jump over corrupt index records and pages, which helps in dumping tables.

- 2 (SRV_FORCE_NO_BACKGROUND)

  Prevent the main thread from running. If a crash would occur during the purge operation, this recovery value prevents it.

- 3 (SRV_FORCE_NO_TRX_UNDO)

  Do not run transaction rollbacks after recovery.

- 4 (SRV_FORCE_NO_IBUF_MERGE)

  Prevent insert buffer merge operations. If they would cause a crash, do not do them. Do not calculate table statistics.

- 5 (SRV_FORCE_NO_UNDO_LOG_SCAN)

  Do not look at undo logs when starting the database: InnoDB treats even incomplete transactions as committed.

- 6 (SRV_FORCE_NO_LOG_REDO)

  Do not do the log roll-forward in connection with recovery.

The database must not otherwise be used with any nonzero value of innodb_force_recovery [1069]. As a safety measure, InnoDB prevents users from performing INSERT, UPDATE, or DELETE operations when innodb_force_recovery [1069] is greater than 0.

Starting from MySQL 3.23.53 and 4.0.4, you can SELECT from tables to dump them, or DROP or CREATE a table even if forced recovery is used. If you know that a certain table is causing a crash in rollback, you can drop it. You can use this also to stop a runaway rollback caused by a failing mass import or ALTER TABLE. You can kill the mysqld process and set innodb_force_recovery [1069] to 3 to bring the database up without the rollback, then DROP the table that is causing the runaway rollback.

13.2.7.3 InnoDB Checkpoints

InnoDB implements a checkpoint mechanism known as “fuzzy” checkpointing. InnoDB flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would in practice stop processing of user SQL statements during the checkpointing process.

During crash recovery, InnoDB looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then InnoDB scans the log files forward from the checkpoint, applying the logged modifications to the database.

InnoDB writes to its log files on a rotating basis. It also writes checkpoint information to the first log file at each checkpoint. All committed modifications that make the database pages in the buffer pool different

from the images on disk must be available in the log files in case InnoDB has to do a recovery. This means that when InnoDB starts to reuse a log file, it has to make sure that the database page images on disk contain the modifications logged in the log file that InnoDB is going to reuse. In other words, InnoDB must create a checkpoint and this often involves flushing of modified database pages to disk.

The preceding description explains why making your log files very large may reduce disk I/O in checkpointing. It often makes sense to set the total size of the log files as large as the buffer pool or even larger. The disadvantage of using large log files is that crash recovery can take longer because there is more logged information to apply to the database.

13.2.8 Moving an InnoDB Database to Another Machine

On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, you should create all databases and tables using lowercase names. A convenient way to accomplish this is to add the following line to the [mysqld] section of your my.cnf or my.ini file before creating any databases or tables:

```
[mysqld]
lower_case_table_names=1
```

Like MyISAM data files, InnoDB data and log files are binary-compatible on all platforms having the same floating-point number format. You can move an InnoDB database simply by copying all the relevant files listed in Section 13.2.7, “Backing Up and Recovering an InnoDB Database”. If the floating-point formats differ but you have not used FLOAT or DOUBLE data types in your tables, then the procedure is the same: simply copy the relevant files. If you use mysqldump to dump your tables on one machine and then import the dump files on the other machine, it does not matter whether the formats differ or your tables contain floating-point data.

One way to increase performance is to switch off autocommit mode when importing data, assuming that the tablespace has enough space for the big rollback segment that the import transactions generate. Do the commit only after importing a whole table or a segment of a table.

13.2.9 The InnoDB Transaction Model and Locking

In the InnoDB transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. InnoDB does locking on the row level and runs queries as nonlocking consistent reads by default, in the style of Oracle. The lock table in InnoDB is stored so space-efficiently that lock escalation is not needed: Typically, several users are permitted to lock every row in InnoDB tables, or any random subset of the rows, without causing InnoDB memory exhaustion.

In InnoDB, all user activity occurs inside a transaction. If autocommit mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts the session for each new connection with autocommit enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 13.2.13, “InnoDB Error Handling”.

A session that has autocommit enabled can perform a multiple-statement transaction by starting it with an explicit START TRANSACTION or BEGIN statement and ending it with a COMMIT or ROLLBACK statement.

If autocommit mode is disabled within a session with SET autocommit = 0, the session always has a transaction open. A COMMIT or ROLLBACK statement ends the current transaction and a new one starts.

A COMMIT means that the changes made in the current transaction are made permanent and become visible to other sessions. A ROLLBACK statement, on the other hand, cancels all modifications made by the current transaction. Both COMMIT and ROLLBACK release all InnoDB locks that were set during the current transaction.

In terms of the SQL:1992 transaction isolation levels, the default InnoDB level is REPEATABLE READ [976]. As of MySQL 4.0.5, InnoDB offers all four transaction isolation levels described by the SQL standard: READ UNCOMMITTED [975], READ COMMITTED [975], REPEATABLE READ [976], and SERIALIZABLE [976]. Before 4.0.5, only REPEATABLE READ [976] and SERIALIZABLE [976] were available. Before MySQL 3.23.50, SET TRANSACTION had no effect on InnoDB tables.

A user can change the isolation level for a single session or for all subsequent connections with the SET TRANSACTION statement. To set the server's default isolation level for all connections, use the --transaction-isolation [394] option on the command line or in an option file. For detailed information about isolation levels and level-setting syntax, see Section 12.3.6, “SET TRANSACTION Syntax”.

In row-level locking, InnoDB normally uses next-key locking. That means that besides index records, InnoDB can also lock the “gap” preceding an index record to block insertions by other sessions in the gap immediately before the index record. A next-key lock refers to a lock that locks an index record and the gap before it. A gap lock refers to a lock that locks only the gap before some index record.

For more information about row-level locking, and the circumstances under which gap locking is disabled, see Section 13.2.9.4, “InnoDB Record, Gap, and Next-Key Locks”.

13.2.9.1 InnoDB Lock Modes

InnoDB implements standard row-level locking where there are two types of locks:

- A shared (*S*) lock permits a transaction to read a row.
- An exclusive (*X*) lock permits a transaction to update or delete a row.

If transaction T1 holds a shared (*S*) lock on row r, then requests from some distinct transaction T2 for a lock on row r are handled as follows:

- A request by T2 for an *S* lock can be granted immediately. As a result, both T1 and T2 hold an *S* lock on r.
- A request by T2 for an *X* lock cannot be granted immediately.

If a transaction T1 holds an exclusive (*X*) lock on row r, a request from some distinct transaction T2 for a lock of either type on r cannot be granted immediately. Instead, transaction T2 has to wait for transaction T1 to release its lock on row r.

Additionally, InnoDB supports *multiple granularity locking* which permits coexistence of record locks and locks on entire tables. To make locking at multiple granularity levels practical, additional types of locks called *intention locks* are used. Intention locks are table locks in InnoDB. The idea behind intention locks is for a transaction to indicate which type of lock (shared or exclusive) it will require later for a row in that table. There are two types of intention locks used in InnoDB (assume that transaction T has requested a lock of the indicated type on table t):

- Intention shared (*IS*): Transaction T intends to set *S* locks on individual rows in table t.
- Intention exclusive (*IX*): Transaction T intends to set *X* locks on those rows.

For example, SELECT ... LOCK IN SHARE MODE sets an *IS* lock and SELECT ... FOR UPDATE sets an *IX* lock.

The intention locking protocol is as follows:

- Before a transaction can acquire an *S* lock on a row in table t, it must first acquire an *IS* or stronger lock on t.

- Before a transaction can acquire an *X* lock on a row, it must first acquire an *IX* lock on `t`.

These rules can be conveniently summarized by means of the following *lock type compatibility matrix*.

	X	***IX***	***S***	***IS***
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks, but not if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Thus, intention locks do not block anything except full table requests (for example, `LOCK TABLES ... WRITE`). The main purpose of *IX* and *IS* locks is to show that someone is locking a row, or going to lock a row in the table.

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an *S* lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+------+
| i    |
+------+
|    1 |
+------+
1 row in set (0.10 sec)
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an *X* lock. The lock cannot be granted because it is incompatible with the *S* lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an *X* lock to delete the row. However, that lock request cannot be granted because client B already has a request for an *X* lock and is waiting for client A to release its *S* lock. Nor can the *S* lock held by A be upgraded to an *X* lock because of the prior request by B for an *X* lock. As a result, InnoDB generates an error for client A and releases its locks. At that point, the lock request for client B can be granted and B deletes the row from the table.

13.2.9.2 Consistent Nonlocking Reads

A consistent read means that InnoDB uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. This exception causes the following anomaly: If you update some rows in a table, a SELECT will see the latest version of the updated rows, but it might also see older versions of any rows. If other sessions simultaneously update the same table, the anomaly means that you may see the table in a state that never existed in the database.

If the transaction isolation level is REPEATABLE READ [976] (the default level), all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

With READ COMMITTED [975] isolation level, each consistent read within a transaction sets and reads its own fresh snapshot.

Consistent read is the default mode in which InnoDB processes SELECT statements in READ COMMITTED [975] and REPEATABLE READ [976] isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other sessions are free to modify those tables at the same time a consistent read is being performed on the table.

Suppose that you are running in the default REPEATABLE READ [976] isolation level. When you issue a consistent read (that is, an ordinary SELECT statement), InnoDB gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

You can advance your timepoint by committing your transaction and then doing another SELECT.

This is called *multi-versioned concurrency control*.

In the following example, session A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

```
             Session A              Session B

           SET autocommit=0;      SET autocommit=0;
time
|          SELECT * FROM t;
|          empty set
|                                  INSERT INTO t VALUES (1, 2);
|
v          SELECT * FROM t;
           empty set
                                  COMMIT;

           SELECT * FROM t;
           empty set

           COMMIT;
```

```
SELECT * FROM t;
---------------------
|    1    |    2    |
---------------------
1 row in set
```

If you want to see the “freshest” state of the database, you should use either the READ COMMITTED [975] isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

With READ COMMITTED [975] isolation level, each consistent read within a transaction sets and reads its own fresh snapshot. With LOCK IN SHARE MODE, a locking read occurs instead: A SELECT blocks until the transaction containing the freshest rows ends (see Section 13.2.9.3, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”).

Consistent read does not work over DROP TABLE or over ALTER TABLE:

- Consistent read does not work over DROP TABLE because MySQL cannot use a table that has been dropped and InnoDB destroys the table.
- Consistent read does not work over ALTER TABLE because ALTER TABLE works by making a temporary copy of the original table and deleting the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken.

13.2.9.3 SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads

In some circumstances, a consistent (nonlocking) read is not convenient and a locking read is required instead. InnoDB supports two types of locking reads:

- SELECT ... LOCK IN SHARE MODE sets a shared mode lock on the rows read. A shared mode lock enables other sessions to read the rows but not to modify them. The rows read are the latest available, so if they belong to another transaction that has not yet committed, the read blocks until that transaction ends.
- For index records the search encounters, SELECT ... FOR UPDATE blocks other sessions from doing SELECT ... LOCK IN SHARE MODE or from reading in certain transaction isolation levels. Consistent reads will ignore any locks set on the records that exist in the read view. (Old versions of a record cannot be locked; they will be reconstructed by applying undo logs on an in-memory copy of the record.)

Locks set by LOCK IN SHARE MODE and FOR UPDATE reads are released when the transaction is committed or rolled back.

As an example of a situation in which a locking read is useful, suppose that you want to insert a new row into a table child, and make sure that the child row has a parent row in table parent. The following discussion describes how to implement referential integrity in application code.

Suppose that you use a consistent read to read the table parent and indeed see the parent row of the to-be-inserted child row in the table. Can you safely insert the child row to table child? No, because it is possible for some other session to delete the parent row from the table parent in the meantime without you being aware of it.

The solution is to perform the SELECT in a locking mode using LOCK IN SHARE MODE:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

A read performed with LOCK IN SHARE MODE reads the latest available data and sets a shared mode lock on the rows read. A shared mode lock prevents others from updating or deleting the row read. Also, if the latest data belongs to a yet uncommitted transaction of another session, we wait until that transaction ends. After we see that the LOCK IN SHARE MODE query returns the parent 'Jones', we can safely add the child record to the child table and commit our transaction.

Let us look at another example: We have an integer counter field in a table child_codes that we use to assign a unique identifier to each child added to table child. It is not a good idea to use either consistent read or a shared mode read to read the present value of the counter because two users of the database may then see the same value for the counter, and a duplicate-key error occurs if two users attempt to add children with the same identifier to the table.

Here, LOCK IN SHARE MODE is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when it attempts to update the counter.

In this case, there are two good ways to implement reading and incrementing the counter:

- First update the counter by incrementing it by 1, and then read it.
- First perform a locking read of the counter using FOR UPDATE, and then increment the counter.

The latter approach can be implemented as follows:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A SELECT ... FOR UPDATE reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL UPDATE would set on the rows.

The preceding description is merely an example of how SELECT ... FOR UPDATE works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The SELECT statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

Note

Locking of rows for update using SELECT FOR UPDATE only applies when autocommit is disabled (either by beginning transaction with START TRANSACTION or by setting autocommit [406] to 0. If autocommit is enabled, the rows matching the specification are not locked.

13.2.9.4 InnoDB Record, Gap, and Next-Key Locks

InnoDB has several types of record-level locks:

- Record lock: This is a lock on an index record.
- Gap lock: This is a lock on a gap between index records, or a lock on the gap before the first or after the last index record.
- Next-key lock: This is a combination of a record lock on the index record and a gap lock on the gap before the index record.

Record locks always lock index records, even if a table is defined with no indexes. For such cases, InnoDB creates a hidden clustered index and uses this index for record locking. See Section 13.2.11.1, “Clustered and Secondary Indexes”.

By default, InnoDB operates in REPEATABLE READ [976] transaction isolation level and with the innodb_locks_unsafe_for_binlog [1070] system variable disabled. In this case, InnoDB uses next-key locks for searches and index scans, which prevents phantom rows (see Section 13.2.9.5, “Avoiding the Phantom Problem Using Next-Key Locking”).

Next-key locking combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order.

Suppose that an index contains the values 10, 11, 13, and 20. The possible next-key locks for this index cover the following intervals, where (or) denote exclusion of the interval endpoint and [or] denote inclusion of the endpoint:

```
(negative infinity, 10]
(10, 11]
(11, 13]
(13, 20]
(20, positive infinity)
```

For the last interval, the next-key lock locks the gap above the largest value in the index and the “supremum” pseudo-record having a value higher than any value actually in the index. The supremum is not a real index record, so, in effect, this next-key lock locks only the gap following the largest index value.

The preceding example shows that a gap might span a single index value, multiple index values, or even be empty.

Gap locking is not needed for statements that lock rows using a unique index to search for a unique row. (This does not include the case that the search condition includes only some columns of a multiple-column unique index; in that case, gap locking does occur.) For example, if the id column has a unique index, the following statement uses only an index-record lock for the row having id value 100 and it does not matter whether other sessions insert rows in the preceding gap:

```
SELECT * FROM child WHERE id = 100;
```

If id is not indexed or has a nonunique index, the statement does lock the preceding gap.

A type of gap lock called an insertion intention gap lock is set by INSERT operations prior to row insertion. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

Gap locking can be disabled explicitly. This occurs if you change the transaction isolation level to READ COMMITTED [975] or enable the innodb_locks_unsafe_for_binlog [1070] system variable. Under these circumstances, gap locking is disabled for searches and index scans and is used only for foreign-key constraint checking and duplicate-key checking.

13.2.9.5 Avoiding the Phantom Problem Using Next-Key Locking

The so-called *phantom* problem occurs within a transaction when the same query produces different sets of rows at different times. For example, if a SELECT is executed twice, but returns a row the second time that was not returned the first time, the row is a “phantom” row.

Suppose that there is an index on the id column of the child table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where id is bigger than 100. Let the table contain rows having id values of 90 and 102. If the locks set on the index records in the scanned range do not lock out inserts made in the gaps (in this case, the gap between 90 and 102), another session can insert a new row into the table with an id of 101. If you were to execute the same SELECT within the same transaction, you would see a new row with an id of 101 (a “phantom”) in the result set returned by the query. If we regard a set of rows as a data item, the new phantom child would violate the isolation principle of transactions that a transaction should be able to run so that the data it has read does not change during the transaction.

To prevent phantoms, InnoDB uses an algorithm called *next-key locking* that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order.

When InnoDB scans an index, it can also lock the gap after the last record in the index. Just that happens in the preceding example: To prevent any insert into the table where id would be bigger than 100, the locks set by InnoDB include a lock on the gap following id value 102.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking enables you to “lock” the nonexistence of something in your table.

Gap locking can be disabled as discussed in Section 13.2.9.4, “InnoDB Record, Gap, and Next-Key Locks”. This may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled.

13.2.9.6 Locks Set by Different SQL Statements in InnoDB

A locking read, an UPDATE, or a DELETE generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter whether there are WHERE conditions in the statement that would exclude the row. InnoDB does not remember the exact WHERE condition, but only knows which index ranges were scanned. The locks are normally next-key locks that also block inserts into the “gap” immediately before the record. However, gap locking can be disabled explicitly, which causes next-key locking not to be used. For more information, see Section 13.2.9.4, “InnoDB Record, Gap, and Next-Key Locks”. The transaction isolation level also can affect which locks are set; see Section 12.3.6, “SET TRANSACTION Syntax”.

If a secondary index is used in a search and index record locks to be set are exclusive, InnoDB also retrieves the corresponding clustered index records and sets locks on them.

Differences between shared and exclusive locks are described in Section 13.2.9.1, “InnoDB Lock Modes”.

If you have no indexes suitable for your statement and MySQL must scan the entire table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily scan many rows.

For SELECT ... FOR UPDATE or SELECT ... LOCK IN SHARE MODE, locks are acquired for scanned rows, and expected to be released for rows that do not qualify for inclusion in the result set (for example, if they do not meet the criteria given in the WHERE clause). However, in some cases, rows might not be unlocked immediately because the relationship between a result row and its original source is lost during query execution. For example, in a UNION, scanned (and locked) rows from a table might be inserted into a temporary table before evaluation whether they qualify for the result set. In this circumstance, the relationship of the rows in the temporary table to the rows in the original table is lost and the latter rows are not unlocked until the end of query execution.

InnoDB sets specific types of locks as follows.

- SELECT ... FROM is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to SERIALIZABLE [976]. For SERIALIZABLE [976] level, the search sets shared next-key locks on the index records it encounters.
- SELECT ... FROM ... LOCK IN SHARE MODE sets shared next-key locks on all index records the search encounters.
- For index records the search encounters, SELECT ... FROM ... FOR UPDATE blocks other sessions from doing SELECT ... FROM ... LOCK IN SHARE MODE or from reading in certain transaction isolation levels. Consistent reads will ignore any locks set on the records that exist in the read view.
- UPDATE ... WHERE ... sets an exclusive next-key lock on every record the search encounters.
- DELETE FROM ... WHERE ... sets an exclusive next-key lock on every record the search encounters.
- INSERT sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next-key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into the gap before the inserted row.

  Prior to inserting the row, a type of gap lock called an insertion intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

  If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same row if another session already has an exclusive lock. This can occur if another session deletes the row. Suppose that an InnoDB table t1 has the following structure:

```
CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
```

  Now suppose that three sessions perform the following operations in order:

  Session 1:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
ROLLBACK;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 rolls back, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

A similar situation occurs if the table already contains a row with key value 1 and three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
COMMIT;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 commits, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

- `INSERT ... ON DUPLICATE KEY UPDATE` differs from a simple `INSERT` in that an exclusive next-key lock rather than a shared lock is placed on the row to be updated when a duplicate-key error occurs.
- `REPLACE` is done like an `INSERT` if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row to be replaced.

- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive index record without a gap lock on each row inserted into `T`. It does the search on rows from `S` as a consistent read (no locks), but sets shared next-key locks on `S` if MySQL binary logging is turned on. `InnoDB` has to set locks in the latter case: In roll-forward recovery from a backup, every SQL statement must be executed in exactly the same way it was done originally.

  `CREATE TABLE ... SELECT ...` performs the `SELECT` with shared next-key locks or as a consistent read, as for `INSERT ... SELECT`.

  For `REPLACE INTO T SELECT ... FROM S WHERE ...`, `InnoDB` sets shared next-key locks on rows from `S`.

- While initializing a previously specified `AUTO_INCREMENT` column on a table, `InnoDB` sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. In accessing the auto-increment counter, `InnoDB` uses a specific `AUTO-INC` table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other sessions cannot insert into the table while the `AUTO-INC` table lock is held; see Section 13.2.9, “The InnoDB Transaction Model and Locking”.

  Before MySQL 3.23.50, `SHOW TABLE STATUS` applied to a table with an `AUTO_INCREMENT` column sets an exclusive row-level lock to the high end of the `AUTO_INCREMENT` index. This means also that `SHOW TABLE STATUS` could cause a deadlock of transactions, something that may surprise users. Starting from MySQL 3.23.50, `InnoDB` fetches the value of a previously initialized `AUTO_INCREMENT` column without setting any locks.

- If a `FOREIGN KEY` constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. `InnoDB` also sets these locks in the case where the constraint fails.

- `LOCK TABLES` sets table locks, but it is the higher MySQL layer above the `InnoDB` layer that sets these locks. Beginning with MySQL 4.0.20 and 4.1.2, `InnoDB` is aware of table locks if `innodb_table_locks = 1` (the default) and `autocommit = 0` [406], and the MySQL layer above `InnoDB` knows about row-level locks.

  Otherwise, InnoDB's automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because in this case the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another session currently has row-level locks. However, this does not endanger transaction integrity, as discussed in Section 13.2.9.8, “Deadlock Detection and Rollback”. See also Section 13.2.15, “Restrictions on InnoDB Tables”.

13.2.9.7 Implicit Transaction Commit and Rollback

By default, MySQL starts the session for each new connection with autocommit mode enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 13.2.13, “InnoDB Error Handling”.

If a session that has autocommit disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.

Some statements implicitly end a transaction, as if you had done a `COMMIT` before executing the statement. For details, see Section 12.3.3, “Statements That Cause an Implicit Commit”.

13.2.9.8 Deadlock Detection and Rollback

`InnoDB` automatically detects transaction deadlocks and rolls back a transaction or transactions to break the deadlock. Starting from MySQL 4.0.5, `InnoDB` tries to pick small transactions to roll back, the size of a

transaction being determined by the number of rows inserted, updated, or deleted. Prior to 4.0.5, InnoDB always rolled back the transaction whose lock request was the last one to build a deadlock, that is, a cycle in the “waits-for” graph of transactions.

Beginning with MySQL 4.0.20 and 4.1.2, InnoDB is aware of table locks if innodb_table_locks = 1 (the default) and autocommit = 0 [406], and the MySQL layer above InnoDB knows about row-level locks. Before that, InnoDB cannot detect deadlocks where a table lock set by a MySQL LOCK TABLES statement is involved, or if a lock set by another storage engine than InnoDB is involved. You have to resolve these situations by setting the value of the innodb_lock_wait_timeout [1069] system variable.

When InnoDB performs a complete rollback of a transaction, all locks set by the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the statement may be preserved. This happens because InnoDB stores row locks in a format such that it cannot know afterward which lock was set by which statement.

13.2.9.9 How to Cope with Deadlocks

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really “atomic”; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- Use SHOW INNODB STATUS to determine the cause of the latest deadlock. That can help you to tune your application to avoid deadlocks. This strategy can be used as of MySQL 3.23.52 and 4.0.3, depending on your MySQL series. From 4.1.2 on, use SHOW ENGINE INNODB STATUS.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Commit your transactions often. Small transactions are less prone to collision.
- If you are using locking reads (SELECT ... FOR UPDATE or SELECT ... LOCK IN SHARE MODE), try using a lower isolation level such as READ COMMITTED [975].
- Access your tables and rows in a fixed order. Then transactions form well-defined queues and do not deadlock.
- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use EXPLAIN SELECT to determine which indexes the MySQL server regards as the most appropriate for your queries.
- Use less locking. If you can afford to permit a SELECT to return data from an old snapshot, do not add the clause FOR UPDATE or LOCK IN SHARE MODE to it. Using the READ COMMITTED [975] isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing else helps, serialize your transactions with table-level locks. The correct way to use LOCK TABLES with transactional tables, such as InnoDB tables, is to begin a transaction with SET autocommit = 0 (not START TRANSACTION) followed by LOCK TABLES, and to not call UNLOCK TABLES until you commit the transaction explicitly. For example, if you need to write to table t1 and read from table t2, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

Table-level locks make your transactions queue nicely and avoid deadlocks.

- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the InnoDB instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.
- In applications that use `autocommit = 1` [406] and MySQL's `LOCK TABLES` statement, InnoDB's internal table locks that were present from 4.0.20 to 4.0.23 can cause deadlocks. Starting from 4.0.22, you can set `innodb_table_locks = 0` in `my.cnf` to fall back to the old behavior and remove the problem. 4.0.24 does not set InnoDB table locks if `autocommit = 1` [406].

13.2.10 InnoDB Multi-Versioning

Because InnoDB is a multi-versioned storage engine, it must keep information about old versions of rows in the tablespace. This information is stored in a data structure called a *rollback segment* (after an analogous data structure in Oracle).

Internally, InnoDB adds three fields to each row stored in the database. A 6-byte `DB_TRX_ID` field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte `DB_ROLL_PTR` field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated. A 6-byte `DB_ROW_ID` field contains a row ID that increases monotonically as new rows are inserted. If InnoDB generates a clustered index automatically, the index contains row ID values. Otherwise, the `DB_ROW_ID` column does not appear in any index.

InnoDB uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which InnoDB has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

You must remember to commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, InnoDB cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space need for your rollback segment.

In the InnoDB multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. Only when InnoDB can discard the update undo log record

written for the deletion can it also physically remove the corresponding row and its index records from the database. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

In a scenario where the user inserts and deletes rows in smallish batches at about the same rate in the table, it is possible that the purge thread starts to lag behind, and the table grows bigger and bigger, making everything disk-bound and very slow. Even if the table would carry just 10MB of useful data, it may grow to occupy 10GB with all the dead rows. In such a case, it would be good to throttle new row operations and allocate more resources for the purge thread. Starting with MySQL 4.0.22 and 4.1.6, the `innodb_max_purge_lag` [1071] system variable exists for exactly this purpose. See Section 13.2.4, "InnoDB Startup Options and System Variables", for more information.

13.2.11 InnoDB Table and Index Structures

MySQL stores its data dictionary information for tables in `.frm` files in database directories. This is true for all MySQL storage engines, but every `InnoDB` table also has its own entry in the `InnoDB` internal data dictionary inside the tablespace. When MySQL drops a table or a database, it has to delete one or more `.frm` files as well as the corresponding entries inside the `InnoDB` data dictionary. Consequently, you cannot move `InnoDB` tables between databases simply by moving the `.frm` files. It is also the reason why `DROP DATABASE` did not work for `InnoDB` type tables before MySQL 3.23.44.

13.2.11.1 Clustered and Secondary Indexes

Every `InnoDB` table has a special index called the *clustered index* where the data for the rows is stored:

- If you define a `PRIMARY KEY` on your table, `InnoDB` uses it as the clustered index.
- If you do not define a `PRIMARY KEY` for your table, MySQL picks the first `UNIQUE` index that has only `NOT NULL` columns as the primary key and `InnoDB` uses it as the clustered index.
- If the table has no `PRIMARY KEY` or suitable `UNIQUE` index, `InnoDB` internally generates a hidden clustered index on a synthetic column containing row ID values. The rows are ordered by the ID that `InnoDB` assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

Accessing a row through the clustered index is fast because the row data is on the same page where the index search leads. If a table is large, the clustered index architecture often saves a disk I/O operation when compared to storage organizations that store row data using a different page from the index record. (For example, `MyISAM` uses one file for data rows and another for index records.)

In `InnoDB`, the records in nonclustered indexes (also called secondary indexes) contain the primary key columns for the row that are not in the secondary index. `InnoDB` uses this primary key value to search for the row in the clustered index. If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

13.2.11.2 Physical Structure of an Index

All `InnoDB` indexes are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. When new records are inserted, `InnoDB` tries to leave 1/16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the fill factor of an index page drops below 1/2, `InnoDB` tries to contract the index tree to free the page.

Note

Changing the page size is not a supported operation and there is no guarantee that InnoDB will function normally with a page size other than 16KB. Problems compiling or running InnoDB may occur.

A version of InnoDB built for one page size cannot use data files or log files from a version built for a different page size.

13.2.11.3 Insert Buffering

It is a common situation in database applications that the primary key is a unique identifier and new rows are inserted in the ascending order of the primary key. Thus, insertions into the clustered index do not require random reads from a disk.

On the other hand, secondary indexes are usually nonunique, and insertions into secondary indexes happen in a relatively random order. This would cause a lot of random disk I/O operations without a special mechanism used in InnoDB.

If an index record should be inserted into a nonunique secondary index, InnoDB checks whether the secondary index page is in the buffer pool. If that is the case, InnoDB does the insertion directly to the index page. If the index page is not found in the buffer pool, InnoDB inserts the record to a special insert buffer structure. The insert buffer is kept so small that it fits entirely in the buffer pool, and insertions can be done very fast.

Periodically, the insert buffer is merged into the secondary index trees in the database. Often it is possible to merge several insertions into the same page of the index tree, saving disk I/O operations. It has been measured that the insert buffer can speed up insertions into a table up to 15 times.

The insert buffer merging may continue to happen *after* the inserting transaction has been committed. In fact, it may continue to happen after a server shutdown and restart (see Section 13.2.7.2, “Forcing InnoDB Recovery”).

Insert buffer merging may take many hours when many secondary indexes must be updated and many rows have been inserted. During this time, disk I/O will be increased, which can cause significant slowdown on disk-bound queries. Another significant background I/O operation is the purge thread (see Section 13.2.10, “InnoDB Multi-Versioning”).

13.2.11.4 Adaptive Hash Indexes

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use hash indexes. InnoDB has a mechanism that monitors index searches made to the indexes defined for a table. If InnoDB notices that queries could benefit from building a hash index, it does so automatically.

The hash index is always built based on an existing B-tree index on the table. InnoDB can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches that InnoDB observes for the B-tree index. A hash index can be partial: It is not required that the whole B-tree index is cached in the buffer pool. InnoDB builds hash indexes on demand for those pages of the index that are often accessed.

In a sense, InnoDB tailors itself through the adaptive hash index mechanism to ample main memory, coming closer to the architecture of main-memory databases.

13.2.11.5 Physical Row Structure

Rows in InnoDB tables have the following characteristics:

- Each index record contains a six-byte header. The header is used to link together consecutive records, and also in row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a six-byte transaction ID field and a seven-byte roll pointer field.
- If no primary key was defined for a table, each clustered index record also contains a six-byte row ID field.
- Each secondary index record also contains all the primary key fields defined for the clustered index key that are not in the secondary index.
- A record contains a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of these pointers is called the record directory. The area where these pointers point is called the data part of the record.
- Internally, InnoDB stores fixed-length character columns such as CHAR(10) in a fixed-length format. InnoDB truncates trailing spaces from VARCHAR columns. Note that MySQL may internally convert CHAR columns to VARCHAR. See Section 12.1.5.2, “Silent Column Specification Changes”.
- An SQL NULL value reserves one or two bytes in the record directory. Besides that, an SQL NULL value reserves zero bytes in the data part of the record if stored in a variable length column. In a fixed-length column, it reserves the fixed length of the column in the data part of the record. Reserving the fixed space for NULL values enables an update of the column from NULL to a non-NULL value to be done in place without causing fragmentation of the index page.

13.2.12 InnoDB Disk I/O and File Space Management

13.2.12.1 InnoDB Disk I/O

InnoDB uses simulated asynchronous disk I/O: InnoDB creates a number of threads to take care of I/O operations, such as read-ahead.

There are two read-ahead heuristics in InnoDB:

- In sequential read-ahead, if InnoDB notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

InnoDB uses a novel file flush technique called *doublewrite*. It adds safety to recovery following an operating system crash or a power outage, and improves performance on most varieties of Unix by reducing the need for fsync() operations.

Doublewrite means that before writing pages to a data file, InnoDB first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write, InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

13.2.12.2 File Space Management

The data files that you define in the configuration file form the InnoDB tablespace. The files are logically concatenated to form the tablespace. There is no striping in use. Currently, you cannot define where within the tablespace your tables are allocated. However, in a newly created tablespace, InnoDB allocates space starting from the first data file.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of size 1MB (64 consecutive pages). The “files” inside a tablespace are called *segments* in InnoDB. The term “rollback segment” is somewhat confusing because it actually contains many tablespace segments.

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it individually. After that, InnoDB starts to allocate whole extents to the segment. InnoDB can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

Two segments are allocated for each index in InnoDB. One is for nonleaf nodes of the B-tree, the other is for the leaf nodes. The idea here is to achieve better sequentiality for the leaf nodes, which contain the data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a SHOW TABLE STATUS statement, InnoDB reports the extents that are definitely free in the tablespace. InnoDB always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only in an (automatic) purge operation after they are no longer needed for transaction rollbacks or consistent reads. (See Section 13.2.10, “InnoDB Multi-Versioning”.)

To see information about the tablespace, use the Tablespace Monitor. See Section 13.2.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”.

The maximum row length, except for variable-length columns (VARBINARY, VARCHAR, BLOB and TEXT), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including BLOB and TEXT columns, must be less than 4GB.

If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page. For a column chosen for off-page storage, InnoDB stores the first 768 bytes locally in the row, and the rest externally into overflow pages. Each such column has its own list of overflow pages. The 768-byte prefix is accompanied by a 20-byte value that stores the true length of the column and points into the overflow list where the rest of the value is stored.

13.2.12.3 Defragmenting a Table

If there are random insertions into or deletions from the indexes of a table, the indexes may become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

One symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All InnoDB data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

(In the preceding query, we are “fooling” the SQL optimizer into scanning the clustered index rather than a secondary index.) Most disks can read 10MB/s to 50MB/s, which can be used to estimate how fast a table scan should be.

It can speed up index scans if you periodically perform a “null” ALTER TABLE operation, which causes MySQL to rebuild the table:

```
ALTER TABLE tbl_name TYPE=InnoDB;
```

Another way to perform a defragmentation operation is to use mysqldump to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions into an index are always ascending and records are deleted only from the end, the InnoDB filespace management algorithm guarantees that fragmentation in the index does not occur.

13.2.13 InnoDB Error Handling

Error handling in InnoDB is not always the same as specified in the SQL standard. According to the standard, any error during an SQL statement should cause rollback of that statement. InnoDB sometimes rolls back only part of the statement, or the whole transaction. The following items describe how InnoDB performs error handling:

- If you run out of file space in the tablespace, a MySQL Table is full error occurs and InnoDB rolls back the SQL statement.
- A transaction deadlock or a timeout in a lock wait causes InnoDB to roll back the whole transaction.

  Both deadlocks and lock wait timeouts are normal on busy servers and it is necessary for applications to be aware that they may happen and handle them by retrying. You can make them less likely by doing as little work as possible between the first change to data during a transaction and the commit, so the locks are held for the shortest possible time and for the smallest possible number of rows. Sometimes splitting work between different transactions may be practical and helpful.

  When a transaction rollback occurs due to a deadlock or lock wait timeout, it cancels the effect of the statements within the transaction. But if the start-transaction statement was START TRANSACTION or BEGIN statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of COMMIT, ROLLBACK, or some SQL statement that causes an implicit commit.
- A duplicate-key error rolls back the SQL statement, if you have not specified the IGNORE option in your statement.
- A row too long error rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code (above the InnoDB storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During such implicit rollbacks, as well as during the explicit ROLLBACK SQL statement, SHOW PROCESSLIST displays "Rolling back" in the State column for the connection (starting from MySQL 4.1.8).

13.2.13.1 InnoDB Error Codes

The following is a nonexhaustive list of common InnoDB-specific errors that you may encounter, with information about why each occurs and how to resolve the problem.

- 1005 (ER_CANT_CREATE_TABLE)

  Cannot create table. If the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers to error –1, table creation probably failed because the table includes a column name that matched the name of an internal InnoDB table.

- 1016 (ER_CANT_OPEN_FILE)

  Cannot find the InnoDB table from the InnoDB data files, although the .frm file for the table exists. See Section 13.2.14.4, “Troubleshooting InnoDB Data Dictionary Operations”.

- 1114 (ER_RECORD_FILE_FULL)

  InnoDB has run out of free space in the tablespace. You should reconfigure the tablespace to add a new data file.

- 1205 (ER_LOCK_WAIT_TIMEOUT)

  Lock wait timeout expired. Transaction was rolled back.

- 1206 (ER_LOCK_TABLE_FULL)

  The total number of locks exceeds the lock table size. To avoid this error, increase the value of innodb_buffer_pool_size [1067]. Within an individual application, a workaround may be to break a large operation into smaller pieces. For example, if the error occurs for a large INSERT, perform several smaller INSERT operations.

- 1213 (ER_LOCK_DEADLOCK)

  Transaction deadlock. You should rerun the transaction.

- 1216 (ER_NO_REFERENCED_ROW)

  You are trying to add a row but there is no parent row, and a foreign key constraint fails. You should add the parent row first.

- 1217 (ER_ROW_IS_REFERENCED)

  You are trying to delete a parent row that has children, and a foreign key constraint fails. You should delete the children first.

13.2.13.2 Operating System Error Codes

To print the meaning of an operating system error number, use the perror program that comes with the MySQL distribution.

The following table provides a list of some common Linux system error codes. For a more complete list, see Linux source code.

- 1 (EPERM)

  Operation not permitted

- 2 (ENOENT)

  No such file or directory

- 3 (ESRCH)

No such process

- 4 (EINTR)

  Interrupted system call

- 5 (EIO)

  I/O error

- 6 (ENXIO)

  No such device or address

- 7 (E2BIG)

  Arg list too long

- 8 (ENOEXEC)

  Exec format error

- 9 (EBADF)

  Bad file number

- 10 (ECHILD)

  No child processes

- 11 (EAGAIN)

  Try again

- 12 (ENOMEM)

  Out of memory

- 13 (EACCES)

  Permission denied

- 14 (EFAULT)

  Bad address

- 15 (ENOTBLK)

  Block device required

- 16 (EBUSY)

  Device or resource busy

- 17 (EEXIST)

  File exists

- 18 (EXDEV)

Cross-device link

- 19 (ENODEV)

  No such device

- 20 (ENOTDIR)

  Not a directory

- 21 (EISDIR)

  Is a directory

- 22 (EINVAL)

  Invalid argument

- 23 (ENFILE)

  File table overflow

- 24 (EMFILE)

  Too many open files

- 25 (ENOTTY)

  Inappropriate ioctl for device

- 26 (ETXTBSY)

  Text file busy

- 27 (EFBIG)

  File too large

- 28 (ENOSPC)

  No space left on device

- 29 (ESPIPE)

  Illegal seek

- 30 (EROFS)

  Read-only file system

- 31 (EMLINK)

  Too many links

The following table provides a list of some common Windows system error codes. For a complete list, see the Microsoft Web site.

- 1 (ERROR_INVALID_FUNCTION)

  Incorrect function.

- 2 (ERROR_FILE_NOT_FOUND)

  The system cannot find the file specified.

- 3 (ERROR_PATH_NOT_FOUND)

  The system cannot find the path specified.

- 4 (ERROR_TOO_MANY_OPEN_FILES)

  The system cannot open the file.

- 5 (ERROR_ACCESS_DENIED)

  Access is denied.

- 6 (ERROR_INVALID_HANDLE)

  The handle is invalid.

- 7 (ERROR_ARENA_TRASHED)

  The storage control blocks were destroyed.

- 8 (ERROR_NOT_ENOUGH_MEMORY)

  Not enough storage is available to process this command.

- 9 (ERROR_INVALID_BLOCK)

  The storage control block address is invalid.

- 10 (ERROR_BAD_ENVIRONMENT)

  The environment is incorrect.

- 11 (ERROR_BAD_FORMAT)

  An attempt was made to load a program with an incorrect format.

- 12 (ERROR_INVALID_ACCESS)

  The access code is invalid.

- 13 (ERROR_INVALID_DATA)

  The data is invalid.

- 14 (ERROR_OUTOFMEMORY)

  Not enough storage is available to complete this operation.

- 15 (ERROR_INVALID_DRIVE)

  The system cannot find the drive specified.

- 16 (ERROR_CURRENT_DIRECTORY)

  The directory cannot be removed.

- 17 (ERROR_NOT_SAME_DEVICE)

  The system cannot move the file to a different disk drive.

- 18 (ERROR_NO_MORE_FILES)

  There are no more files.

- 19 (ERROR_WRITE_PROTECT)

  The media is write protected.

- 20 (ERROR_BAD_UNIT)

  The system cannot find the device specified.

- 21 (ERROR_NOT_READY)

  The device is not ready.

- 22 (ERROR_BAD_COMMAND)

  The device does not recognize the command.

- 23 (ERROR_CRC)

  Data error (cyclic redundancy check).

- 24 (ERROR_BAD_LENGTH)

  The program issued a command but the command length is incorrect.

- 25 (ERROR_SEEK)

  The drive cannot locate a specific area or track on the disk.

- 26 (ERROR_NOT_DOS_DISK)

  The specified disk or diskette cannot be accessed.

- 27 (ERROR_SECTOR_NOT_FOUND)

  The drive cannot find the sector requested.

- 28 (ERROR_OUT_OF_PAPER)

  The printer is out of paper.

- 29 (ERROR_WRITE_FAULT)

  The system cannot write to the specified device.

- 30 (ERROR_READ_FAULT)

  The system cannot read from the specified device.

- 31 (ERROR_GEN_FAILURE)

  A device attached to the system is not functioning.

- 32 (ERROR_SHARING_VIOLATION)

The process cannot access the file because it is being used by another process.

- 33 (ERROR_LOCK_VIOLATION)

  The process cannot access the file because another process has locked a portion of the file.

- 34 (ERROR_WRONG_DISK)

  The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.

- 36 (ERROR_SHARING_BUFFER_EXCEEDED)

  Too many files opened for sharing.

- 38 (ERROR_HANDLE_EOF)

  Reached the end of the file.

- 39 (ERROR_HANDLE_DISK_FULL)

  The disk is full.

- 87 (ERROR_INVALID_PARAMETER)

  The parameter is incorrect. (If this error occurs on MySQL 4.1.9 on Windows and you have set innodb_file_per_table [1068] in a server option file, this is Bug #8021, and a workaround is to add the line innodb_flush_method=unbuffered to the file as well.)

- 112 (ERROR_DISK_FULL)

  The disk is full.

- 123 (ERROR_INVALID_NAME)

  The file name, directory name, or volume label syntax is incorrect.

- 1450 (ERROR_NO_SYSTEM_RESOURCES)

  Insufficient system resources exist to complete the requested service.

13.2.14 InnoDB Performance Tuning and Troubleshooting

13.2.14.1 InnoDB Performance Tuning Tips

The followings tips are grouped by category. Some of them can apply in multiple categories, so it is useful to read them all.

Storage Layout Tips

- In InnoDB, having a long PRIMARY KEY wastes a lot of disk space because its value must be stored with every secondary index record. (See Section 13.2.11, “InnoDB Table and Index Structures”.) Create an AUTO_INCREMENT column as the primary key if your primary key is long.

- Use the VARCHAR data type instead of CHAR if you are storing variable-length strings or if the column may contain many NULL values. A CHAR(*N*) column always takes *N* characters to store data, even if the string is shorter or its value is NULL. Smaller tables fit better in the buffer pool and reduce disk I/O.

Transaction Management Tips

- Wrap several modifications into a single transaction to reduce the number of flush operations. InnoDB must flush the log to disk at each transaction commit if that transaction made modifications to the database. The rotation speed of a disk is typically at most 167 revolutions/second (for a 10,000RPM disk), which constrains the number of commits to the same 167th of a second if the disk does not "fool" the operating system.
- If you can afford the loss of some of the latest committed transactions if a crash occurs, you can set the innodb_flush_log_at_trx_commit [1068] parameter to 0. InnoDB tries to flush the log once per second anyway, although the flush is not guaranteed.

Disk I/O Tips

- innodb_buffer_pool_size [1067] specifies the size of the buffer pool. If your buffer pool is small and you have sufficient memory, making the pool larger can improve performance by reducing the amount of disk I/O needed as queries access InnoDB tables. For more information about the pool, see Section 7.5.2, "The InnoDB Buffer Pool".
- Beware of big rollbacks of mass inserts: InnoDB uses the insert buffer to save disk I/O in inserts, but no such mechanism is used in a corresponding rollback. A disk-bound rollback can take 30 times as long to perform as the corresponding insert. Killing the database process does not help because the rollback starts again on server startup. The only way to get rid of a runaway rollback is to increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or to use a special procedure. See Section 13.2.7.2, "Forcing InnoDB Recovery".
- Beware also of other big disk-bound operations. Use DROP TABLE and CREATE TABLE to empty a table, not DELETE FROM *tbl_name*.
- (Relevant from 3.23.39 up.) In some versions of GNU/Linux and Unix, flushing files to disk with the Unix fsync() call (which InnoDB uses by default) and other similar methods is surprisingly slow. If you are dissatisfied with database write performance, you might try setting the innodb_flush_method [1069] parameter to O_DSYNC. The O_DSYNC flush method seems to perform slower on most systems, but yours might not be one of them.
- (Verified using MySQL 4.1, assumed for other MySQL versions, given that this is a platform architecture issue.) When using the InnoDB storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), it is important to use direct I/O for InnoDB-related files. Failure to do so may cause degradation of InnoDB's speed and performance on this platform. To use direct I/O for an entire UFS file system used for storing InnoDB-related files, mount it with the forcedirectio option; see mount_ufs(1M). (The default on Solaris 10/x86_64 is *not* to use this option.)

  When using the InnoDB storage engine with a large innodb_buffer_pool_size [1067] value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), a significant performance gain might be achieved by placing InnoDB data files and log files on raw devices or on a separate direct I/O UFS file system using the forcedirectio mount option as described earlier. Users of the Veritas file system VxFS should use the convosync=direct mount option. You are advised to perform tests with and without raw partitions or direct I/O file systems to verify whether performance is improved on your system.

  Other MySQL data files, such as those for MyISAM tables, should not be placed on a direct I/O file system. Executables or libraries *must not* be placed on a direct I/O file system.
- If the Unix top tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound. Maybe you are making too many transaction commits, or the buffer pool is too small. Making the buffer pool bigger can help, but do not set it equal to more than 80% of physical memory.

Logging Tips

- Make your log files big, even as big as the buffer pool. When InnoDB has written the log files full, it must write the modified contents of the buffer pool to disk in a checkpoint. Small log files cause many unnecessary disk writes. The disadvantage of big log files is that the recovery time is longer.
- Make the log buffer quite large as well (on the order of 8MB).

Bulk Data Loading Tips

- When importing data into InnoDB, make sure that MySQL does not have autocommit mode enabled because that requires a log flush to disk for every insert. To disable autocommit during your import operation, surround it with SET autocommit and COMMIT statements:

```
SET autocommit=0;
... SQL import statements ...
COMMIT;
```

  If you use the mysqldump option --opt [297], you get dump files that are fast to import into an InnoDB table, even without wrapping them with the SET autocommit and COMMIT statements.

- If you have UNIQUE constraints on secondary keys, starting from MySQL 3.23.52 and 4.0.3, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

```
SET unique_checks=0;
... SQL import statements ...
SET unique_checks=1;
```

  For big tables, this saves a lot of disk I/O because InnoDB can use its insert buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys.

- If you have FOREIGN KEY constraints in your tables, starting from MySQL 3.23.52 and 4.0.3, you can speed up table imports by turning the foreign key checks off for a while in the import session:

```
SET foreign_key_checks=0;
... SQL import statements ...
SET foreign_key_checks=1;
```

  For big tables, this can save a lot of disk I/O.

Other Tips

- Unlike MyISAM, InnoDB does not store an index cardinality value in its tables. Instead, InnoDB computes a cardinality for a table the first time it accesses it after startup. With a large number of tables, this might take significant time. It is the initial table open operation that is important, so to “warm up” a table for later use, access it immediately after startup by issuing a statement such as SELECT 1 FROM *tbl_name* LIMIT 1.
- Use the multiple-row INSERT syntax to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

  This tip is valid for inserts into any table, not just InnoDB tables.

- If you often have recurring queries for tables that are not updated frequently, enable the query cache (available as of MySQL 4.0):

```
[mysqld]
query_cache_type = 1
query_cache_size = 10M
```

In MySQL 4.0, the query cache works only with autocommit enabled. This restriction is removed in MySQL 4.1.1 and up.

13.2.14.2 SHOW ENGINE INNODB STATUS and the InnoDB Monitors

Starting from MySQL 3.23.42, InnoDB Monitors provide information about the InnoDB internal state. This information is useful for performance tuning. Each Monitor can be enabled by creating a table with a special name, which causes InnoDB to write Monitor output periodically. Also, starting from MySQL 4.1.2, output for the standard InnoDB Monitor is available on demand using the SHOW ENGINE INNODB STATUS SQL statement. (From MySQL 3.23.52 and 4.0.3 until 4.1.1, you can use the SHOW INNODB STATUS SQL statement.)

There are several types of InnoDB Monitors:

- The standard InnoDB Monitor displays the following types of information:
  - Table and record locks held by each active transaction
  - Lock waits of a transactions
  - Semaphore waits of threads
  - Pending file I/O requests
  - Buffer pool statistics
  - Purge and insert buffer merge activity of the main InnoDB thread

  For a discussion of InnoDB lock modes, see Section 13.2.9.1, “InnoDB Lock Modes”.

  To enable the standard InnoDB Monitor for periodic output, create a table named innodb_monitor. To obtain Monitor output on demand, use the SHOW ENGINE INNODB STATUS SQL statement to fetch the output to your client program. If you are using the mysql interactive client, the output is more readable if you replace the usual semicolon statement terminator with \G:

```
mysql> SHOW ENGINE INNODB STATUS\G
```

- The InnoDB Lock Monitor is like the standard Monitor but also provides extensive lock information. To enable this Monitor for periodic output, create a table named innodb_lock_monitor.
- The InnoDB Tablespace Monitor prints a list of file segments in the shared tablespace and validates the tablespace allocation data structures. To enable this Monitor for periodic output, create a table named innodb_tablespace_monitor.
- The InnoDB Table Monitor prints the contents of the InnoDB internal data dictionary. To enable this Monitor for periodic output, create a table named innodb_table_monitor. The Table Monitor is available as of MySQL 3.23.44.

To enable an InnoDB Monitor for periodic output, use a CREATE TABLE statement to create the table associated with the Monitor. For example, to enable the standard InnoDB Monitor, create the innodb_monitor table:

```
CREATE TABLE innodb_monitor (a INT) TYPE=INNODB;
```

To stop the Monitor, drop the table:

```
DROP TABLE innodb_monitor;
```

The CREATE TABLE syntax is just a way to pass a command to the InnoDB engine through MySQL's SQL parser: The only things that matter are the table name innodb_monitor and that it be an InnoDB table. The structure of the table is not relevant at all for the InnoDB Monitor. If you shut down the server, the Monitor does not restart automatically when you restart the server. You must drop the Monitor table and issue a new CREATE TABLE statement to start the Monitor. (This syntax may change in a future release.)

When you enable InnoDB Monitors for periodic output, InnoDB writes their output to the mysqld server standard error output (stderr). In this case, no output is sent to clients. When switched on, InnoDB Monitors print data about every 15 seconds. Server output usually is directed to the error log (see Section 5.3.1, "The Error Log"). This data is useful in performance tuning. On Windows, you must start the server from a command prompt in a console window with the --console [385] option if you want to direct the output to the window rather than to the error log.

Beginning with MySQL 4.0.19, InnoDB sends diagnostic output to stderr or files instead of stdout or fixed-size memory buffers, to avoid potential buffer overflow errors. As a side effect, the output of SHOW INNODB STATUS is written to a status file in the MySQL data directory every fifteen seconds. The name of the file is innodb_status.*pid*, where *pid* is the server process ID. InnoDB removes the file for a normal shutdown. If abnormal shutdowns have occurred, instances of these status files may be present and must be removed manually. Before removing them, you might want to examine them to see whether they contain useful information about the cause of abnormal shutdowns. Beginning with MySQL 4.0.21, the innodb_status.*pid* file is created only if the configuration option innodb-status-file=1 [1066] is set.

InnoDB Monitors should be enabled only when you actually want to see Monitor information because output generation does result in some performance decrement. Also, if you enable monitor output by creating the associated table, your error log may become quite large if you forget to remove the table later.

For additional information about InnoDB monitors, see the following resources:

- Mark Leith: InnoDB Table and Tablespace Monitors
- MySQL Performance Blog: SHOW INNODB STATUS walk through

Each monitor begins with a header containing a timestamp and the monitor name. For example:

```
=====================================
090407 12:06:19 INNODB TABLESPACE MONITOR OUTPUT
=====================================
```

The header for the standard Monitor (INNODB MONITOR OUTPUT) is also used for the Lock Monitor because the latter produces the same output with the addition of extra lock information.

The following sections describe the output for each Monitor.

InnoDB Standard Monitor and Lock Monitor Output

The Lock Monitor is the same as the standard Monitor except that it includes additional lock information. Enabling either monitor for periodic output by creating the associated InnoDB table turns on the same output stream, but the stream includes the extra information if the Lock Monitor is enabled. For example, if you create the innodb_monitor and innodb_lock_monitor tables, that turns on a single output stream. The stream includes extra lock information until you disable the Lock Monitor by removing the innodb_lock_monitor table.

Example InnoDB Monitor output:

```
mysql> SHOW INNODB STATUS\G
*************************** 1. row ***************************
Status:
=====================================
030709 13:00:59 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 18 seconds
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the
semaphore: X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 108462, OS waits 37964; RW-excl spins 681824, OS waits
375485
------------------------
LATEST FOREIGN KEY ERROR
------------------------
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id 34831
inserting
15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
Foreign key constraint fails for table test/ibtest11a:
,
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b` (`A`,
  `D`) ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
 0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2:
 len 4; hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ...N..;; 4:
 len 7; hex 00000000c3e0a7; asc .......;; 5: len 4; hex 6b68446b; asc khDk;;
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits 0 0: len 4; hex 8000015b; asc ...[;; 1: len 4; hex
80000005; asc ....;; 2: len 3; hex 6b6864; asc khd;; 3: len 6; hex
0000111ef3eb; asc ......;; 4: len 7; hex 800001001e0084; asc .......;; 5:
len 3; hex 6b6864; asc khd;;
------------------------
LATEST DETECTED DEADLOCK
------------------------
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185, OS thread id 30733
inserting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794,'aA35818','bb','c79166','d4766t',
'e187358f','g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d
%H:%i'),7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
symbole trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138;
asc aa35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190, OS thread id 32782
inserting
130 lock struct(s), heap size 11584, undo log entries 437
```

```
MySQL thread id 23, query id 4554396 localhost heikki update
REPLACE INTO alex1 VALUES(NULL, 32, NULL,'aa3572','','c3572','d6012t','',
NULL,'h396', NULL, NULL, 7.31,7.31,7.31,200)
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
symbole trx id 0 290251546 lock_mode X locks rec but not gap
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138;
asc aa35818;; 1:
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
symbole trx id 0 290251546 lock_mode X locks gap before rec insert intention
waiting
Record lock, heap no 82 RECORD: info bits 0 0: len 7; hex 61613335373230;
asc aa35720;; 1:
*** WE ROLL BACK TRANSACTION (1)
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
Total number of lock structs in row lock hash table 70
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3491, OS thread id 42002
MySQL thread id 32, query id 4668737 localhost heikki
show innodb status
---TRANSACTION 0 290328384, ACTIVE 0 sec, process no 3205, OS thread id
38929 inserting
1 lock struct(s), heap size 320
MySQL thread id 29, query id 4668736 localhost heikki update
insert into speedc values (1519229,1, 'hgjhjgghggjgjgjgjgjggjgjgjgjgjgggjgjg
jlhhgghgggggghhjhghgggggggjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjfh
---TRANSACTION 0 290328383, ACTIVE 0 sec, process no 3180, OS thread id
28684 committing
1 lock struct(s), heap size 320, undo log entries 1
MySQL thread id 19, query id 4668734 localhost heikki update
insert into speedcm values (1603393,1, 'hgjhjgghggjgjgjgjgjggjgjgjgjgjggggjgj
gjlhhgghgggggghhjhghgggggggjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjf
---TRANSACTION 0 290328327, ACTIVE 0 sec, process no 3200, OS thread id
36880 starting index read
LOCK WAIT 2 lock struct(s), heap size 320
MySQL thread id 27, query id 4668644 localhost heikki Searching rows for
update
update ibtest11a set B = 'kHdkkkk' where A = 89572
------- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 65556 n bits 232 table test/ibtest11a index
PRIMARY trx id 0 290328327 lock_mode X waiting
Record lock, heap no 1 RECORD: info bits 0 0: len 9; hex 73757072656d756d00;
asc supremum.;;
------------------
---TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id
34831 rollback of SQL statement
ROLLING BACK 14 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
---TRANSACTION 0 290327208, ACTIVE 1 sec, process no 3190, OS thread id
32782
58 lock struct(s), heap size 5504, undo log entries 159
MySQL thread id 23, query id 4668732 localhost heikki update
REPLACE INTO alex1 VALUES(86, 46, 538,'aa95666','bb','c95666','d9486t',
'e200498f','g86814','h538',date_format('2001-04-03 12:54:22','%Y-%m-%d
%H:%i'),
---TRANSACTION 0 290323325, ACTIVE 3 sec, process no 3185, OS thread id
30733 inserting
4 lock struct(s), heap size 1024, undo log entries 165
MySQL thread id 21, query id 4668735 localhost heikki update
INSERT INTO alex1 VALUES(NULL, 49, NULL,'aa42837','','c56319','d1719t','',
```

```
NULL,'h321', NULL, NULL, 7.31,7.31,7.31,200)
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s
---
LOG
---
Log sequence number 18 1212842764
Log flushed up to   18 1212665295
Last checkpoint at  18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 84966343; in additional pool allocated 1402624
Buffer pool size   3200
Free buffers       110
Database pages     3074
Modified db pages  2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

`InnoDB` Monitor output is limited to 64,000 bytes when produced using the `SHOW ENGINE INNODB STATUS` statement. This limit does not apply to output written to the server's error output.

Some notes on the output sections:

SEMAPHORES

This section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside `InnoDB`. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting the `innodb_thread_concurrency` [1072] system variable smaller than the default value might help in such situations.

LATEST FOREIGN KEY ERROR

This section provides information about the most recent foreign key constraint error. It is not present if no such error has occurred. The contents include the statement that failed as well as information about the constraint that failed and the referenced and referencing tables.

LATEST DETECTED DEADLOCK

This section provides information about the most recent deadlock. It is not present if no deadlock has occurred. The contents show which transactions are involved, the statement each was attempting to execute, the locks they have and need, and which transaction InnoDB decided to roll back to break the deadlock. The lock modes reported in this section are explained in Section 13.2.9.1, "InnoDB Lock Modes".

TRANSACTIONS

If this section reports lock waits, your applications might have lock contention. The output can also help to trace the reasons for transaction deadlocks.

FILE I/O

This section provides information about threads that InnoDB uses to perform various types of I/O. The first few of these are dedicated to general InnoDB processing. The contents also display information for pending I/O operations and statistics for I/O performance.

On Unix, the number of threads is always 4. On Windows, the number depends on the setting of the innodb_file_io_threads [1068] system variable.

INSERT BUFFER AND ADAPTIVE HASH INDEX

This section shows the status of the InnoDB insert buffer and adaptive hash index. (See Section 13.2.11.3, "Insert Buffering", and Section 13.2.11.4, "Adaptive Hash Indexes".) The contents include the number of operations performed for each, plus statistics for hash index performance.

LOG

This section displays information about the InnoDB log. The contents include the current log sequence number, how far the log has been flushed to disk, and the position at which InnoDB last took a checkpoint. (See Section 13.2.7.3, "InnoDB Checkpoints".) The section also displays information about pending writes and write performance statistics.

BUFFER POOL AND MEMORY

This section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.

For additional information about the operation of the buffer pool, see Section 7.5.2, "The InnoDB Buffer Pool".

ROW OPERATIONS

This section shows what the main thread is doing, including the number and performance rate for each type of row operation.

InnoDB Tablespace Monitor Output

The InnoDB Tablespace Monitor prints information about the file segments in the shared tablespace and validates the tablespace allocation data structures. If you use individual tablespaces by enabling innodb_file_per_table [1068], the Tablespace Monitor does not describe those tablespaces.

Example InnoDB Tablespace Monitor output:

```
=============================================
090408 21:28:09 INNODB TABLESPACE MONITOR OUTPUT
=============================================
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
SEGMENT id 0 2 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 3 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
SEGMENT id 0 488 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 17 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 171 space 0; page 2; res 592 used 481; full ext 7
fragm pages 16; free extents 0; not full extents 2: pages 17
SEGMENT id 0 172 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 173 space 0; page 2; res 96 used 44; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 12
...
SEGMENT id 0 601 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
NUMBER of file segments: 73
Validating tablespace
Validation ok
---------------------------------------
END OF INNODB TABLESPACE MONITOR OUTPUT
=======================================
```

The Tablespace Monitor output includes information about the shared tablespace as a whole, followed by a list containing a breakdown for each segment within the tablespace.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of size 1MB (64 consecutive pages).

The initial part of the output that displays overall tablespace information has this format:

```
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
```

Overall tablespace information includes these values:

- `id`: The tablespace ID. A value of 0 refers to the shared tablespace.
- `size`: The current tablespace size in pages.
- `free limit`: The minimum page number for which the free list has not been initialized. Pages at or above this limit are free.
- `free extents`: The number of free extents.

- `not full frag extents`, `used pages`: The number of fragment extents that are not completely filled, and the number of pages in those extents that have been allocated.
- `full frag extents`: The number of completely full fragment extents.
- `first seg id not used`: The first unused segment ID.

Individual segment information has this format:

```
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
```

Segment information includes these values:

`id`: The segment ID.

`space`, `page`: The tablespace number and page within the tablespace where the segment “inode” is located. A tablespace number of 0 indicates the shared tablespace. `InnoDB` uses inodes to keep track of segments in the tablespace. The other fields displayed for a segment (`id`, `res`, and so forth) are derived from information in the inode.

`res`: The number of pages allocated (reserved) for the segment.

`used`: The number of allocated pages in use by the segment.

`full ext`: The number of extents allocated for the segment that are completely used.

`fragm pages`: The number of initial pages that have been allocated to the segment.

`free extents`: The number of extents allocated for the segment that are completely unused.

`not full extents`: The number of extents allocated for the segment that are partially used.

`pages`: The number of pages used within the not-full extents.

When a segment grows, it starts as a single page, and `InnoDB` allocates the first pages for it individually, up to 32 pages (this is the `fragm pages` value). After that, `InnoDB` allocates complete 64-page extents. `InnoDB` can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

For the example segment shown earlier, it has 32 fragment pages, plus 2 full extents (64 pages each), for a total of 160 pages used out of 160 pages allocated. The following segment has 32 fragment pages and one partially full extent using 14 pages for a total of 46 pages used out of 96 pages allocated:

```
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
```

It is possible for a segment that has extents allocated to it to have a `fragm pages` value less than 32 if some of the individual pages have been deallocated subsequent to extent allocation.

`InnoDB` Table Monitor Output

The `InnoDB` Table Monitor prints the contents of the `InnoDB` internal data dictionary.

The output contains one section per table. The `SYS_FOREIGN` and `SYS_FOREIGN_COLS` sections are for internal data dictionary tables that maintain information about foreign keys. There are also sections for the Table Monitor table and each user-created `InnoDB` table. Suppose that the following two tables have been created in the `test` database:

```
CREATE TABLE parent
```

```
(
  par_id    INT NOT NULL,
  fname      CHAR(20),
  lname      CHAR(20),
  PRIMARY KEY (par_id),
  UNIQUE INDEX (lname, fname)
) ENGINE = INNODB;

CREATE TABLE child
(
  par_id      INT NOT NULL,
  child_id    INT NOT NULL,
  name        VARCHAR(40),
  birth       DATE,
  weight      DECIMAL(10,2),
  misc_info   VARCHAR(255),
  last_update TIMESTAMP,
  PRIMARY KEY (par_id, child_id),
  INDEX (name),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = INNODB;
```

Then the Table Monitor output will look something like this (reformatted slightly):

```
===========================================
090420 12:04:38 INNODB TABLE MONITOR OUTPUT
===========================================
--------------------------------------
TABLE: name SYS_FOREIGN, id 0 11, columns 8, indexes 3, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           FOR_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           REF_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           N_COLS: DATA_INT len 4 prec 0;
           DB_ROW_ID: DATA_SYS DATA_ROW_ID len 6 prec 0;
           DB_TRX_ID: DATA_SYS DATA_TRX_ID len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS DATA_ROLL_PTR len 7 prec 0;
  INDEX: name ID_IND, id 0 11, fields 1/6, type 3
   root page 46, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID DB_TRX_ID DB_ROLL_PTR FOR_NAME REF_NAME N_COLS
  INDEX: name FOR_IND, id 0 12, fields 1/2, type 0
   root page 47, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  FOR_NAME ID
  INDEX: name REF_IND, id 0 13, fields 1/2, type 0
   root page 48, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  REF_NAME ID
--------------------------------------
TABLE: name SYS_FOREIGN_COLS, id 0 12, columns 8, indexes 1, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           POS: DATA_INT len 4 prec 0;
           FOR_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           REF_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0 prec 0;
           DB_ROW_ID: DATA_SYS DATA_ROW_ID len 6 prec 0;
           DB_TRX_ID: DATA_SYS DATA_TRX_ID len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS DATA_ROLL_PTR len 7 prec 0;
  INDEX: name ID_IND, id 0 14, fields 2/6, type 3
   root page 49, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID POS DB_TRX_ID DB_ROLL_PTR FOR_COL_NAME REF_COL_NAME
--------------------------------------
TABLE: name test/child, id 0 14, columns 11, indexes 2, appr.rows 238
  COLUMNS: par_id: DATA_INT len 4 prec 0;
           child_id: DATA_INT len 4 prec 0;
           name: DATA_VARCHAR prtype 1 len 40 prec 0;
           birth: DATA_INT len 3 prec 0;
           weight: type 11 len 12 prec 0;
```

```
           misc_info: DATA_VARCHAR prtype 1 len 255 prec 0;
           last_update: DATA_INT len 4 prec 0;
           DB_ROW_ID: DATA_SYS DATA_ROW_ID len 6 prec 0;
           DB_TRX_ID: DATA_SYS DATA_TRX_ID len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS DATA_ROLL_PTR len 7 prec 0;
  INDEX: name PRIMARY, id 0 17, fields 2/9, type 3
   root page 52, appr.key vals 238, leaf pages 6, size pages 7
   FIELDS:  par_id child_id DB_TRX_ID DB_ROLL_PTR name birth weight misc_info last_update
  INDEX: name name, id 0 18, fields 1/3, type 0
   root page 53, appr.key vals 210, leaf pages 1, size pages 1
   FIELDS:  name par_id child_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
             REFERENCES test/parent ( par_id )
-------------------------------------
TABLE: name test/innodb_table_monitor, id 0 15, columns 5, indexes 1, appr.rows 0
  COLUMNS: i: DATA_INT len 4 prec 0;
           DB_ROW_ID: DATA_SYS DATA_ROW_ID len 6 prec 0;
           DB_TRX_ID: DATA_SYS DATA_TRX_ID len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS DATA_ROLL_PTR len 7 prec 0;
  INDEX: name GEN_CLUST_INDEX, id 0 19, fields 0/4, type 1
   root page 56, appr.key vals 0, leaf pages 1, size pages 1
   FIELDS:  DB_ROW_ID DB_TRX_ID DB_ROLL_PTR i
-------------------------------------
TABLE: name test/parent, id 0 13, columns 7, indexes 2, appr.rows 223
  COLUMNS: par_id: DATA_INT len 4 prec 0;
           fname: DATA_CHAR prtype 2 len 20 prec 0;
           lname: DATA_CHAR prtype 2 len 20 prec 0;
           DB_ROW_ID: DATA_SYS DATA_ROW_ID len 6 prec 0;
           DB_TRX_ID: DATA_SYS DATA_TRX_ID len 6 prec 0;
           DB_ROLL_PTR: DATA_SYS DATA_ROLL_PTR len 7 prec 0;
  INDEX: name PRIMARY, id 0 15, fields 1/5, type 3
   root page 50, appr.key vals 223, leaf pages 2, size pages 3
   FIELDS:  par_id DB_TRX_ID DB_ROLL_PTR fname lname
  INDEX: name lname, id 0 16, fields 2/3, type 2
   root page 51, appr.key vals 300, leaf pages 1, size pages 1
   FIELDS:  lname fname par_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
             REFERENCES test/parent ( par_id )
-----------------------------------
END OF INNODB TABLE MONITOR OUTPUT
===================================
```

For each table, Table Monitor output contains a section that displays general information about the table and specific information about its columns, indexes, and foreign keys.

The general information for each table includes the table name (in *`db_name/tbl_name`* format except for internal tables), its ID, the number of columns and indexes, and an approximate row count.

The `COLUMNS` part of a table section lists each column in the table. Information for each column indicates its name and data type characteristics. Some internal columns are added by `InnoDB`, such as `DB_ROW_ID` (row ID), `DB_TRX_ID` (transaction ID), and `DB_ROLL_PTR` (a pointer to the rollback/undo data).

- `DATA_`*`xxx`*: These symbols indicate the data type. There may be multiple `DATA_`*`xxx`* symbols for a given column.
- `prtype`: The column's “precise” type. This field includes information such as the column data type, character set code, nullability, signedness, and whether it is a binary string. This field is described in the `innobase/include/data0type.h` source file.
- `len`: The column length in bytes.
- `prec`: The precision of the type.

Each `INDEX` part of the table section provides the name and characteristics of one table index:

- name: The index name. If the name is PRIMARY, the index is a primary key. If the name is GEN_CLUST_INDEX, the index is the clustered index that is created automatically if the table definition doesn't include a primary key or non-NULL unique index. See Section 13.2.11.1, “Clustered and Secondary Indexes”.
- id: The index ID.
- fields: The number of fields in the index, as a value in *m*/*n* format:
  - *m* is the number of user-defined columns; that is, the number of columns you would see in the index definition in a CREATE TABLE statement.
  - *n* is the total number of index columns, including those added internally. For the clustered index, the total includes the other columns in the table definition, plus any columns added internally. For a secondary index, the total includes the columns from the primary key that are not part of the secondary index.
- type: The index type. This is a bit field. For example, 1 indicates a clustered index and 2 indicates a unique index, so a clustered index (which always contains unique values), will have a type value of 3. An index with a type value of 0 is neither clustered nor unique. The flag values are defined in the innobase/include/dict0mem.h source file.
- root page: The index root page number.
- appr. key vals: The approximate index cardinality.
- leaf pages: The approximate number of leaf pages in the index.
- size pages: The approximate total number of pages in the index.
- FIELDS: The names of the fields in the index. For a clustered index that was generated automatically, the field list begins with the internal DB_ROW_ID (row ID) field. DB_TRX_ID and DB_ROLL_PTR are always added internally to the clustered index, following the fields that comprise the primary key. For a secondary index, the final fields are those from the primary key that are not part of the secondary index.

The end of the table section lists the FOREIGN KEY definitions that apply to the table. This information appears whether the table is a referencing or referenced table.

13.2.14.3 InnoDB General Troubleshooting

The following general guidelines apply to troubleshooting InnoDB problems:

- When an operation fails or you suspect a bug, you should look at the MySQL server error log (see Section 5.3.1, “The Error Log”).
- Issues relating to the InnoDB data dictionary include failed CREATE TABLE statements (orphaned table files), inability to open .InnoDB files, and system cannot find the path specified errors. For information about these sorts of problems and errors, see Section 13.2.14.4, “Troubleshooting InnoDB Data Dictionary Operations”.
- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through mysqld_safe or as a Windows service. You can then see what mysqld prints to the console, and so have a better grasp of what is going on. On Windows, start mysqld with the --console [385] option to direct the output to the console window.
- Use the InnoDB Monitors to obtain information about a problem (see Section 13.2.14.2, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”). If the problem is performance-related, or your

server appears to be hung, you should use the standard Monitor to print information about the internal state of InnoDB. If the problem is with locks, use the Lock Monitor. If the problem is in creation of tables or other data dictionary operations, use the Table Monitor to print the contents of the InnoDB internal data dictionary. To see tablespace information use the Tablespace Monitor.

- If you suspect that a table is corrupt, run CHECK TABLE on that table.

13.2.14.4 Troubleshooting InnoDB Data Dictionary Operations

A specific issue with tables is that the MySQL server keeps data dictionary information in .frm files it stores in the database directories, whereas InnoDB also stores the information into its own data dictionary inside the tablespace files. If you move .frm files around, or use DROP DATABASE in MySQL versions before 3.23.44, or the server crashes in the middle of a data dictionary operation, the locations of the .frm files may end up out of synchrony with the locations recorded in the InnoDB internal data dictionary.

A symptom of an out-of-sync data dictionary is that a CREATE TABLE statement fails. If this occurs, you should look in the server's error log. If the log says that the table already exists inside the InnoDB internal data dictionary, you have an orphaned table inside the InnoDB tablespace files that has no corresponding .frm file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphaned table by following the instructions given in the error message. If you are still unable to use DROP TABLE successfully, the problem may be due to name completion in the mysql client. To work around this problem, start the mysql client with the --skip-auto-rehash [260] option and try DROP TABLE again. (With name completion on, mysql tries to construct a list of table names, which fails when a problem such as just described exists.)

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open a .InnoDB file:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphaned .frm file without a corresponding table inside InnoDB. You can drop the orphaned .frm file by deleting it manually.

If MySQL crashes in the middle of an ALTER TABLE operation, you may end up with an orphaned temporary table inside the InnoDB tablespace. Using the Table Monitor, you can see listed a table with a name that begins with #sql-.... Starting from MySQL 4.0.6, you can perform SQL statements also on tables whose name contains the character "#" if you enclose the name within backticks. Thus, you can drop such an orphaned table like any other orphaned table using the method described earlier. To copy

or rename a file in the Unix shell, you need to put the file name in double quotation marks if the file name contains "#".

Older MySQL versions did not permit accessing any table with a name containing "#". The solution in older MySQL versions is to use a special InnoDB mechanism available starting from MySQL 3.23.48. When you have an orphaned table #sql-id inside the tablespace, you can cause InnoDB to rename it to rsql-id_recover_innodb_tmp_table with the following statement:

```
CREATE TABLE `rsql-id_recover_innodb_tmp_table`(...) TYPE=InnoDB;
```

With innodb_file_per_table [1068] enabled, the following message might occur if the .frm or .ibd files (or both) are missing:

```
InnoDB: in InnoDB data dictionary has tablespace id N,
InnoDB: but tablespace with that id or name does not exist. Have
InnoDB: you deleted or moved .ibd files?
InnoDB: This may also be a table created with CREATE TEMPORARY TABLE
InnoDB: whose .ibd and .frm files MySQL automatically removed, but the
InnoDB: table still exists in the InnoDB internal data dictionary.
```

If this occurs, try the following procedure to resolve the problem:

1. Create a matching .frm file in some other database directory and copy it to the database directory where the orphan table is located.
2. Issue DROP TABLE for the original table. That should successfully drop the table and InnoDB should print a warning to the error log that the .ibd file was missing.

13.2.15 Restrictions on InnoDB Tables

Warning

Do *not* convert MySQL system tables in the mysql database from MyISAM to InnoDB tables! This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or re-generate them with the mysql_install_db script.

Warning

It is not a good idea to configure InnoDB to use data files or log files on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

- A table cannot contain more than 1000 columns.
- The InnoDB internal maximum key length is 3500 bytes, but MySQL itself restricts this to 1024 bytes.
- Index key prefixes can be up to 767 bytes (255 bytes before MySQL 4.1.2). See Section 12.1.4, "CREATE INDEX Syntax".
- The maximum row length, except for variable-length columns (VARBINARY, VARCHAR, BLOB and TEXT), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including BLOB and TEXT columns, must be less than 4GB.

  If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page, as described in Section 13.2.12.2, "File Space Management".

- On some older operating systems, files must be less than 2GB. This is not a limitation of InnoDB itself, but if you require a large tablespace, you will need to configure it using several smaller data files rather than one or a file large data files.
- The combined size of the InnoDB log files must be less than 4GB.
- The minimum tablespace size is 10MB. The maximum tablespace size is four billion database pages (64TB). This is also the maximum size for a table.
- InnoDB tables do not support FULLTEXT indexes.
- InnoDB tables do not support spatial data types.
- ANALYZE TABLE determines index cardinality (as displayed in the Cardinality column of SHOW INDEX output) by doing eight random dives to each of the index trees and updating index cardinality estimates accordingly. Because these are only estimates, repeated runs of ANALYZE TABLE may produce different numbers. This makes ANALYZE TABLE fast on InnoDB tables but not 100% accurate because it does not take all rows into account.

  MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you can try using ANALYZE TABLE. In the few cases that ANALYZE TABLE does not produce values good enough for your particular tables, you can use FORCE INDEX with your queries to force the use of a particular index, or set the max_seeks_for_key [420] system variable to ensure that MySQL prefers index lookups over table scans. See Section 5.1.3, “Server System Variables”, and Section B.5.6, “Optimizer-Related Issues”.
- SHOW TABLE STATUS does not give accurate statistics on InnoDB tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.
- InnoDB does not keep an internal count of rows in a table. (In practice, this would be somewhat complicated due to multi-versioning.) To process a SELECT COUNT(*) FROM t statement, InnoDB must scan an index of the table, which takes some time if the index is not entirely in the buffer pool. If your table does not change often, using the MySQL query cache is a good solution. To get a fast count, you have to use a counter table you create yourself and let your application update it according to the inserts and deletes it does. SHOW TABLE STATUS also can be used if an approximate row count is sufficient. See Section 13.2.14.1, “InnoDB Performance Tuning Tips”.
- On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, you should create all databases and tables using lowercase names.
- For an AUTO_INCREMENT column, you must always define an index for the table, and that index must contain just the AUTO_INCREMENT column. In MyISAM tables, the AUTO_INCREMENT column may be part of a multi-column index.
- Before MySQL 4.1.12, InnoDB does not support the AUTO_INCREMENT table option for setting the initial sequence value in an ALTER TABLE statement. Before MySQL 4.1.14, the same is true for CREATE TABLE. To set the value with InnoDB, insert a dummy row with a value one less and delete that dummy row, or insert the first row with an explicit value specified.
- While initializing a previously specified AUTO_INCREMENT column on a table, InnoDB sets an exclusive lock on the end of the index associated with the AUTO_INCREMENT column. In accessing the auto-increment counter, InnoDB uses a specific table lock mode AUTO-INC where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other clients cannot insert into the table while the AUTO-INC table lock is held; see Section 13.2.5.3, “AUTO_INCREMENT Handling in InnoDB”.

- When you restart the MySQL server, InnoDB may reuse an old value that was generated for an AUTO_INCREMENT column but never stored (that is, a value that was generated during an old transaction that was rolled back).
- When an AUTO_INCREMENT column runs out of values, InnoDB wraps a BIGINT to -9223372036854775808 and BIGINT UNSIGNED to 1. However, BIGINT values have 64 bits, so if you were to insert one million rows per second, it would still take nearly three hundred thousand years before BIGINT reached its upper bound. With all other integer type columns, a duplicate-key error results. This is similar to how MyISAM works, because it is mostly general MySQL behavior and not about any storage engine in particular.
- DELETE FROM *tbl_name* does not regenerate the table but instead deletes all rows, one by one.
- Under some conditions, TRUNCATE *tbl_name* for an InnoDB table is mapped to DELETE FROM *tbl_name* and does not reset the AUTO_INCREMENT counter. See Section 12.1.10, "TRUNCATE TABLE Syntax".
- Before MySQL 4.0.14 or 4.1.0, if you tried to create a unique index on a prefix of a column you got an error:

  ```
  CREATE TABLE T (A CHAR(20), B INT, UNIQUE (A(5))) TYPE = InnoDB;
  ```

  If you created a nonunique index on a prefix of a column, InnoDB created an index over the whole column. These restrictions were removed in MySQL 4.0.14.
- Before MySQL 4.0.20 or 4.1.2, the MySQL LOCK TABLES operation does not know about InnoDB row-level locks set by completed SQL statements. This means that you can get a table lock on a table even if there still exist transactions by other users who have row-level locks on the same table. Thus, your operations on the table may have to wait if they collide with these locks of other users. Also a deadlock is possible. However, this does not endanger transaction integrity, because the row-level locks set by InnoDB always take care of the integrity. Also, a table lock prevents other transactions from acquiring more row-level locks (in a conflicting lock mode) on the table.
- Beginning with MySQL 4.0.20 and 4.1.2, the MySQL LOCK TABLES operation acquires two locks on each table if innodb_table_locks=1 (the default). In addition to a table lock on the MySQL layer, it also acquires an InnoDB table lock. Older versions of MySQL do not acquire InnoDB table locks. Beginning with MySQL 4.0.22 and 4.1.7, the old behavior can be selected by setting innodb_table_locks=0. If no InnoDB table lock is acquired, LOCK TABLES completes even if some records of the tables are being locked by other transactions.
- All InnoDB locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke LOCK TABLES on InnoDB tables in autocommit = 1 [406] mode, because the acquired InnoDB table locks would be released immediately.
- Sometimes it would be useful to lock further tables in the course of a transaction. Unfortunately, LOCK TABLES in MySQL performs an implicit COMMIT and UNLOCK TABLES. An InnoDB variant of LOCK TABLES has been planned that can be executed in the middle of a transaction.
- Before MySQL 3.23.52, replication always ran with autocommit enabled. Therefore consistent reads in the slave would also see partially processed transactions, and thus the read would not be really consistent in the slave. This restriction was removed in MySQL 3.23.52.
- The LOAD TABLE FROM MASTER statement for setting up replication slave servers does not work for InnoDB tables. A workaround is to alter the table to MyISAM on the master, then do the load, and after that alter the master table back to InnoDB. Do not do this if the tables use InnoDB-specific features such as foreign keys.

- The default database page size in InnoDB is 16KB. By recompiling the code, you can set it to values ranging from 8KB to 64KB. You must update the values of UNIV_PAGE_SIZE and UNIV_PAGE_SIZE_SHIFT in the univ.i source file.

  Note

  Changing the page size is not a supported operation and there is no guarantee that InnoDB will function normally with a page size other than 16KB. Problems compiling or running InnoDB may occur.

  A version of InnoDB built for one page size cannot use data files or log files from a version built for a different page size.

- You cannot create a table with a column name that matches the name of an internal InnoDB column (including DB_ROW_ID, DB_TRX_ID, DB_ROLL_PTR, and DB_MIX_ID). In versions of MySQL before 4.1.19 this would cause a crash, since 4.1.19 the server will report error 1005 and refers to error –1 in the error message. This limitation applies only to use of the names in uppercase.

- InnoDB has a limit of 1023 concurrent transactions that have created undo records by modifying data. Workarounds include keeping transactions as small and fast as possible and delaying changes until near the end of the transaction. Applications should commit transactions before doing time-consuming client-side operations.

13.3 The MERGE Storage Engine

The MERGE storage engine was introduced in MySQL 3.23.25. It is also known as the MRG_MyISAM engine.

A MERGE table is a collection of identical MyISAM tables that can be used as one. “Identical” means that all tables have identical column and index information. You cannot merge MyISAM tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the MyISAM tables can be compressed with myisampack. See Section 4.6.4, “myisampack — Generate Compressed, Read-Only MyISAM Tables”. Differences in table options such as AVG_ROW_LENGTH, MAX_ROWS, or PACK_KEYS do not matter.

When you create a MERGE table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format, and an .MRG file contains the names of the underlying MyISAM tables that should be used as one. (Originally, all used tables had to be in the same database as the MERGE table. This restriction has been lifted as of MySQL 4.1.1.)

You can use SELECT, DELETE, UPDATE, and (as of MySQL 4.0) INSERT on MERGE tables. You must have SELECT [492], DELETE [491], and UPDATE [493] privileges on the MyISAM tables that you map to a MERGE table.

Note

The use of MERGE tables entails the following security issue: If a user has access to MyISAM table *t*, that user can create a MERGE table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*. If this behavior is undesirable, you can start the server with the new --skip-merge [393] option to disable the MERGE storage engine. This option is available as of MySQL 4.1.21.

Use of DROP TABLE with a MERGE table drops only the MERGE specification. The underlying tables are not affected.

To create a MERGE table, you must specify a UNION=(*list-of-tables*) option that indicates which MyISAM tables to use. You can optionally specify an INSERT_METHOD option to control how inserts into the MERGE table take place. Use a value of FIRST or LAST to cause inserts to be made in the first or last underlying table, respectively. If you specify no INSERT_METHOD option or if you specify it with a value of NO, inserts into the MERGE table are not permitted and attempts to do so result in an error.

The following example shows how to create a MERGE table:

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    ->    a INT NOT NULL AUTO_INCREMENT,
    ->    message CHAR(20), INDEX(a))
    ->    ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term from MySQL 4.0.18 on and TYPE is deprecated.

Note that column a is indexed as a PRIMARY KEY in the underlying MyISAM tables, but not in the MERGE table. There it is indexed but not as a PRIMARY KEY because a MERGE table cannot enforce uniqueness over the set of underlying tables. (Similarly, a column with a UNIQUE index in the underlying tables should be indexed in the MERGE table but not as a UNIQUE index.)

After creating the MERGE table, you can use it to issue queries that operate on the group of tables as a whole:

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

To remap a MERGE table to a different collection of MyISAM tables, you can use one of the following methods:

- DROP the MERGE table and re-create it.
- Use ALTER TABLE *tbl_name* UNION=(...) to change the list of underlying tables.

As of MySQL 4.1.23, the underlying table definitions and indexes must conform more closely than previously to the definition of the MERGE table. Conformance is checked when a table that is part of a MERGE table is opened, not when the MERGE table is created. If any table fails the conformance checks, the operation that triggered the opening of the table fails. This means that changes to the definitions of tables within a MERGE may cause a failure when the MERGE table is accessed. The conformance checks applied to each table are:

- The underlying table and the MERGE table must have the same number of columns.

- The column order in the underlying table and the `MERGE` table must match.
- Additionally, the specification for each corresponding column in the parent `MERGE` table and the underlying tables are compared and must satisfy these checks:
  - The column type in the underlying table and the `MERGE` table must be equal.
  - The column length in the underlying table and the `MERGE` table must be equal.
  - The column of the underlying table and the `MERGE` table can be `NULL`.
- The underlying table must have at least as many indexes as the `MERGE` table. The underlying table may have more indexes than the `MERGE` table, but cannot have fewer.

Note

A known issue exists where indexes on the same columns must be in identical order, in both the `MERGE` table and the underlying `MyISAM` table. See Bug #33653.

Each index must satisfy these checks:

- The index type of the underlying table and the `MERGE` table must be the same.
- The number of index parts (that is, multiple columns within a compound index) in the index definition for the underlying table and the `MERGE` table must be the same.
- For each index part:
  - Index part lengths must be equal.
  - Index part types must be equal.
  - Index part languages must be equal.
  - Check whether index parts can be `NULL`.

For information about the table checks applied prior to MySQL 4.1.23, see Section 13.3.2, “`MERGE` Table Problems”.

Additional Resources

- A forum dedicated to the `MERGE` storage engine is available at http://forums.mysql.com/list.php?93.

13.3.1 `MERGE` Table Advantages and Disadvantages

`MERGE` tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with `myisampack`, and then create a `MERGE` table to use them as one.
- Obtain more speed. You can split a large read-only table based on some criteria, and then put individual tables on different disks. A `MERGE` table structured this way could be much faster than using a single large table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the underlying tables for some queries and use a `MERGE` table for others. You can even have many different `MERGE` tables that use overlapping sets of tables.

- Perform more efficient repairs. It is easier to repair individual smaller tables that are mapped to a MERGE table than to repair a single large table.
- Instantly map many tables as one. A MERGE table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, MERGE table collections are *very* fast to create or remap. (You must still specify the index definitions when you create a MERGE table, even though no indexes are created.)
- If you have a set of tables from which you create a large table on demand, you can instead create a MERGE table from them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each MyISAM table is bound by this limit, but a collection of MyISAM tables is not.
- You can create an alias or synonym for a MyISAM table by defining a MERGE table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and memcpy() calls for each read).

The disadvantages of MERGE tables are:

- You can use only identical MyISAM tables for a MERGE table.
- Some MyISAM features are unavailable in MERGE tables. For example, you cannot create FULLTEXT indexes on MERGE tables. (You can create FULLTEXT indexes on the underlying MyISAM tables, but you cannot search the MERGE table with a full-text search.)
- If the MERGE table is nontemporary, all underlying MyISAM tables must be nontemporary. If the MERGE table is temporary, the MyISAM tables can be any mix of temporary and nontemporary.
- MERGE tables use more file descriptors than MyISAM tables. If 10 clients are using a MERGE table that maps to 10 tables, the server uses (10 × 10) + 10 file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Index reads are slower. When you read an index, the MERGE storage engine needs to issue a read on all underlying tables to check which one most closely matches a given index value. To read the next index value, the MERGE storage engine needs to search the read buffers to find the next value. Only when one index buffer is used up does the storage engine need to read the next index block. This makes MERGE indexes much slower on eq_ref [567] searches, but not much slower on ref [567] searches. For more information about eq_ref [567] and ref [567], see Section 12.7.2, “EXPLAIN Syntax”.

13.3.2 MERGE Table Problems

The following are known problems with MERGE tables:

- If you use ALTER TABLE to change a MERGE table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying MyISAM tables are copied into the altered table, which then uses the specified storage engine.
- Before MySQL 4.1.1, all underlying tables and the MERGE table itself had to be in the same database.
- The INSERT_METHOD table option for a MERGE table indicates which underlying MyISAM table to use for inserts into the MERGE table. However, use of the AUTO_INCREMENT table option for that MyISAM table has no effect for inserts into the MERGE table until at least one row has been inserted directly into the MyISAM table.
- A MERGE table cannot maintain uniqueness constraints over the entire table. When you perform an INSERT, the data goes into the first or last MyISAM table (as determined by the INSERT_METHOD

option). MySQL ensures that unique key values remain unique within that MyISAM table, but not over all the underlying tables in the collection.

- Because the MERGE engine cannot enforce uniqueness over the set of underlying tables, REPLACE does not work as expected. The two key facts are:

  - REPLACE can detect unique key violations only in the underlying table to which it is going to write (which is determined by the INSERT_METHOD option). This differs from violations in the MERGE table itself.

  - If REPLACE detects a unique key violation, it will change only the corresponding row in the underlying table it is writing to; that is, the first or last table, as determined by the INSERT_METHOD option.

  Similar considerations apply for INSERT ... ON DUPLICATE KEY UPDATE.

- You should not use ANALYZE TABLE, REPAIR TABLE, OPTIMIZE TABLE, ALTER TABLE, DROP TABLE, DELETE without a WHERE clause, or TRUNCATE TABLE on any of the tables that are mapped into an open MERGE table. If you do so, the MERGE table may still refer to the original table and yield unexpected results. To work around this problem, ensure that no MERGE tables remain open by issuing a FLUSH TABLES statement prior to performing any of the named operations.

  The unexpected results include the possibility that the operation on the MERGE table will report table corruption. If this occurs after one of the named operations on the underlying MyISAM tables, the corruption message is spurious. To deal with this, issue a FLUSH TABLES statement after modifying the MyISAM tables.

- DROP TABLE on a table that is in use by a MERGE table does not work on Windows because the MERGE storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not permit open files to be deleted, so you first must flush all MERGE tables (with FLUSH TABLES) or drop the MERGE table before dropping the table.

- Before MySQL 3.23.49, DELETE FROM *merge_table* used without a WHERE clause only clears the mapping for the table. That is, it incorrectly empties the .MRG file rather than deleting records from the mapped tables.

- Using RENAME TABLE on an active MERGE table may corrupt the table. This is fixed in MySQL 4.1.x.

- As of MySQL 4.1.23, the definition of the MyISAM tables and the MERGE table are checked when the tables are accessed (for example, as part of a SELECT or INSERT statement). The checks ensure that the definitions of the tables and the parent MERGE table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables, an error is returned and the statement fails. Because these checks take place when the tables are opened, any changes to the definition of a single table, including column changes, column ordering, and engine alterations will cause the statement to fail.

  Prior to MySQL 4.1.23, table checks are applied as follows:

  - When you create or alter MERGE table, there is no check to ensure that the underlying tables are existing MyISAM tables and have identical structures. When the MERGE table is used, MySQL checks that the row length for all mapped tables is equal, but this is not foolproof. If you create a MERGE table from dissimilar MyISAM tables, you are very likely to run into strange problems.

  - Similarly, if you create a MERGE table from non-MyISAM tables, or if you drop an underlying table or alter it to be a non-MyISAM table, no error for the MERGE table occurs until later when you attempt to use it.

  - Because the underlying MyISAM tables need not exist when the MERGE table is created, you can create the tables in any order, as long as you do not use the MERGE table until all of its underlying tables are in place. Also, if you can ensure that a MERGE table will not be used during a given period, you can perform maintenance operations on the underlying tables, such as backing up or restoring them, altering them, or dropping and recreating them. It is not necessary to redefine the MERGE table temporarily to exclude the underlying tables while you are operating on them.

- The order of indexes in the MERGE table and its underlying tables should be the same. If you use ALTER TABLE to add a UNIQUE index to a table used in a MERGE table, and then use ALTER TABLE to add a nonunique index on the MERGE table, the index ordering is different for the tables if there was already a nonunique index in the underlying table. (This happens because ALTER TABLE puts UNIQUE indexes before nonunique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.

- If you encounter an error message similar to `ERROR 1017 (HY000): Can't find file: 'tbl_name.MRG' (errno: 2)`, it generally indicates that some of the underlying tables do not use the MyISAM storage engine. Confirm that all of these tables are MyISAM.

- The maximum number of rows in a MERGE table is 2^{32} (~4.295E+09; the same as for a MyISAM table). It is not possible to merge multiple MyISAM tables into a single MERGE table that would have more than this number of rows.

- The MERGE storage engine does not support INSERT DELAYED statements.

13.4 The MEMORY (HEAP) Storage Engine

The MEMORY storage engine creates tables with contents that are stored in memory. Before MySQL 4.1, MEMORY tables are called HEAP tables. As of 4.1, MEMORY is the preferred term, although HEAP remains supported for backward compatibility.

The MEMORY storage engine associate each table with one disk file. The file name begins with the table name and has an extension of .frm to indicate that it stores the table definition.

To specify that you want to create a MEMORY table, indicate that with an ENGINE table option:

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term from MySQL 4.0.18 on and TYPE is deprecated.

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in .frm files on disk, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test TYPE=MEMORY
    ->     SELECT ip,SUM(downloads) AS down
    ->     FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY tables have the following characteristics:

- Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put

in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.

- MEMORY tables permit up to 32 indexes per table and 16 columns per index. Previously, the maximum key length supported by this storage engine was 255 bytes; as of MySQL 4.1.13, MEMORY tables support a maximum key length of 500 bytes. (See Section C.1.13, “Changes in MySQL 4.1.13 (2005-07-15)”.)

- Before MySQL 4.1, the MEMORY storage engine supports only hash indexes. From MySQL 4.1 on, hash indexes are still the default, but you can specify explicitly that a MEMORY table index should be a HASH or BTREE by adding a USING clause as shown here:

```
CREATE TABLE lookup
    (id INT, INDEX USING HASH (id))
    ENGINE = MEMORY;
CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
```

  For general characteristics of B-tree and hash indexes, see Section 7.4.3, “How MySQL Uses Indexes”.

- If a MEMORY table hash index has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a BTREE index to avoid this problem.

- MEMORY tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)

- As of MySQL 4.0.2, columns that are indexed can contain NULL values.

- MEMORY tables use a fixed-length row-storage format. Variable-length types such as VARCHAR are stored using a fixed length.

- MEMORY tables cannot contain BLOB or TEXT columns.

- MEMORY supports AUTO_INCREMENT columns as of MySQL 4.1.0.

- As of MySQL 4.1, MEMORY supports INSERT DELAYED. See Section 12.2.4.2, “INSERT DELAYED Syntax”.

- Non-TEMPORARY MEMORY tables are shared among all clients, just like any other non-TEMPORARY table.

- MEMORY table contents are stored in memory, which is a property that MEMORY tables share with internal temporary tables that the server creates on the fly while processing queries. However, the two types of tables differ in that MEMORY tables are not subject to storage conversion, whereas internal temporary tables are:

  - MEMORY tables are never converted to disk tables. If an internal temporary table becomes too large, the server automatically converts it to on-disk storage, as described in Section 7.7.4, “How MySQL Uses Internal Temporary Tables”.

  - The maximum size of MEMORY tables is limited by the max_heap_table_size [419] system variable, which has a default value of 16MB. To have larger (or smaller) MEMORY tables, you must change the value of this variable. The value in effect for CREATE TABLE is the value used for the life of the table. (If you use ALTER TABLE or TRUNCATE TABLE, the value in effect at that time becomes the new maximum size for the table. A server restart also sets the maximum size of existing MEMORY tables to the global max_heap_table_size [419] value.) You can set the size for individual tables as described later in this section.

- The server needs sufficient memory to maintain all MEMORY tables that are in use at the same time.
- To free memory used by a MEMORY table when you no longer require its contents, you should execute DELETE or TRUNCATE TABLE to remove all rows, or remove the table altogether using DROP TABLE.
- If you want to populate a MEMORY table when the MySQL server starts, you can use the --init-file [387] option. For example, you can put statements such as INSERT INTO ... SELECT or LOAD DATA INFILE into this file to load the table from a persistent data source. See Section 5.1.2, "Server Command Options", and Section 12.2.5, "LOAD DATA INFILE Syntax".
- A server's MEMORY tables become empty when it is shut down and restarted. However, if the server is a replication master, its slave are not aware that these tables have become empty, so they returns out-of-date content if you select data from these tables. To handle this, as of MySQL 4.0.18, when a MEMORY table is used on a master for the first time since it was started, a DELETE FROM statement is written to the master's binary log automatically, thus synchronizing the slave to the master again. Note that even with this strategy, the slave still has outdated data in the table during the interval between the master's restart and its first use of the table. However, if you use the --init-file [387] option to populate the MEMORY table on the master at startup, it ensures that this time interval is zero.
- The memory needed for one row in a MEMORY table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) × 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) × 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

  ALIGN() represents a round-up factor to cause the row length to be an exact multiple of the char pointer size. sizeof(char*) is 4 on 32-bit machines and 8 on 64-bit machines.

As mentioned earlier, the max_heap_table_size [419] system variable sets the limit on the maximum size of MEMORY tables. To control the maximum size for individual tables, set the session value of this variable before creating each table. (Do not change the global max_heap_table_size [419] value unless you intend the value to be used for MEMORY tables created by all clients.) The following example creates two MEMORY tables, with a maximum size of 1MB and 2MB, respectively:

```
mysql> SET max_heap_table_size = 1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.01 sec)

mysql> SET max_heap_table_size = 1024*1024*2;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.00 sec)
```

Both tables will revert to the server's global max_heap_table_size [419] value if the server restarts.

You can also specify a MAX_ROWS table option in CREATE TABLE statements for MEMORY tables to provide a hint about the number of rows you plan to store in them. This does not enable the table to grow beyond the max_heap_table_size [419] value, which still acts as a constraint on maximum table size. For maximum flexibility in being able to use MAX_ROWS, set max_heap_table_size [419] at least as high as the value to which you want each MEMORY table to be able to grow.

Additional Resources

- A forum dedicated to the MEMORY storage engine is available at http://forums.mysql.com/list.php?92.

13.5 The BDB (BerkeleyDB) Storage Engine

Sleepycat Software has provided MySQL with the Berkeley DB transactional storage engine. This storage engine typically is called BDB for short. BDB tables may have a greater chance of surviving crashes and are also capable of COMMIT and ROLLBACK operations on transactions.

Support for the BDB storage engine is included in MySQL source distributions, which come with a BDB distribution that is patched to make it work with MySQL. You cannot use an unpatched version of BDB with MySQL.

BDB support will be removed

As of MySQL 5.1, BDB is not supported.

For general information about Berkeley DB, please visit the Sleepycat Web site, http://www.sleepycat.com/.

13.5.1 Operating Systems Supported by BDB

Currently, we know that the BDB storage engine works with the following operating systems:

- Linux 2.x Intel
- Sun Solaris (SPARC and x86)
- FreeBSD 4.x/5.x (x86, sparc64)
- IBM AIX 4.3.x
- SCO OpenServer
- SCO UnixWare 7.1.x
- Windows NT/2000/XP

The BDB storage engine does *not* work with the following operating systems:

- Linux 2.x Alpha
- Linux 2.x AMD64
- Linux 2.x IA-64
- Linux 2.x s390
- Mac OS X

Note

The preceding lists are not complete. We update them as we receive more information.

If you build MySQL from source with support for BDB tables, but the following error occurs when you start mysqld, it means that the BDB storage engine is not supported for your architecture:

```
bdb: architecture lacks fast mutexes: applications cannot be threaded
Can't init databases
```

In this case, you must rebuild MySQL without BDB support or start the server with the --skip-bdb [392] option.

13.5.2 Installing BDB

If you have downloaded a binary version of MySQL that includes support for Berkeley DB, simply follow the usual binary distribution installation instructions. (MySQL-Max distributions include BDB support.)

If you build MySQL from source, you can enable BDB support by invoking configure with the --with-berkeley-db option in addition to any other options that you normally use. Download a distribution for MySQL 3.23.34 or newer, change location into its top-level directory, and run this command:

```
shell> ./configure --with-berkeley-db [other-options]
```

For more information, see Section 5.2, “The mysqld-max Extended MySQL Server”, Section 2.8, “Installing MySQL from Generic Binaries on Other Unix-Like Systems”, and Section 2.9, “Installing MySQL from Source”.

13.5.3 BDB Startup Options

The following options to mysqld can be used to change the behavior of the BDB storage engine. For more information, see Section 5.1.2, “Server Command Options”.

- --bdb-home=*path* [1138]

  The base directory for BDB tables. This should be the same directory that you use for --datadir [385].

- --bdb-lock-detect=*method* [1138]

  The BDB lock detection method. The option value should be DEFAULT, OLDEST, RANDOM, or YOUNGEST.

- --bdb-logdir=*file_name* [1138]

  The BDB log file directory.

- --bdb-no-recover [1138]

  Do not start Berkeley DB in recover mode.

- --bdb-no-sync [1138]

  Don't synchronously flush the BDB logs. This option is deprecated as of MySQL 4.0.18; use --skip-sync-bdb-logs [1138] instead (see the description for --sync-bdb-logs [1138]).

- --bdb-shared-data [1138]

  Start Berkeley DB in multi-process mode. (Do not use DB_PRIVATE when initializing Berkeley DB.)

- --bdb-tmpdir=*path* [1138]

  The BDB temporary file directory.

- --skip-bdb [392]

  Disable the BDB storage engine.

- --sync-bdb-logs [1138]

Synchronously flush the BDB logs. This option is enabled by default. Use --skip-sync-bdb-logs [1138] to disable it. This option was added in MySQL 4.0.18.

If you use the --skip-bdb [392] option, MySQL does not initialize the Berkeley DB library and this saves a lot of memory. However, if you use this option, you cannot use BDB tables. If you try to create a BDB table, MySQL uses the default storage engine instead.

Normally, you should start mysqld without the --bdb-no-recover [1138] option if you intend to use BDB tables. However, this may cause problems when you try to start mysqld if the BDB log files are corrupted. See Section 2.10.2.3, "Starting and Troubleshooting the MySQL Server".

With the bdb_max_lock [407] variable, you can specify the maximum number of locks that can be active on a BDB table. The default is 10,000. You should increase this if errors such as the following occur when you perform long transactions or when mysqld has to examine many rows to execute a query:

```
bdb: Lock table is out of available locks
Got error 12 from ...
```

You may also want to change the binlog_cache_size [407] and max_binlog_cache_size [1183] variables if you are using large multiple-statement transactions. See Section 5.3.4, "The Binary Log".

See also Section 5.1.3, "Server System Variables".

13.5.4 Characteristics of BDB Tables

Each BDB table is stored on disk in two files. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format, and a .db file contains the table data and indexes.

To specify explicitly that you want a BDB table, indicate that with an ENGINE table option:

```
CREATE TABLE t (i INT) ENGINE = BDB;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term from MySQL 4.0.18 on and TYPE is deprecated.

BerkeleyDB is a synonym for BDB in the ENGINE table option.

The BDB storage engine provides transactional tables. The way you use these tables depends on the autocommit mode:

- If you are running with autocommit enabled (which is the default), changes to BDB tables are committed immediately and cannot be rolled back.
- If you are running with autocommit disabled, changes do not become permanent until you execute a COMMIT statement. Instead of committing, you can execute ROLLBACK to forget the changes.

  You can start a transaction with the START TRANSACTION or BEGIN statement to suspend autocommit, or with SET autocommit = 0 to disable autocommit explicitly.

For more information about transactions, see Section 12.3.1, "START TRANSACTION, COMMIT, and ROLLBACK Syntax".

The BDB storage engine has the following characteristics:

- BDB tables can have up to 31 indexes per table, 16 columns per index, and a maximum key size of 1024 bytes (500 bytes before MySQL 4.0).

- MySQL requires a primary key in each BDB table so that each row can be uniquely identified. If you don't create one explicitly by declaring a PRIMARY KEY, MySQL creates and maintains a hidden primary key for you. The hidden key has a length of five bytes and is incremented for each insert attempt. This key does not appear in the output of SHOW CREATE TABLE or DESCRIBE.

- The primary key is faster than any other index, because it is stored together with the row data. The other indexes are stored as the key data plus the primary key, so it is important to keep the primary key as short as possible to save disk space and get better speed.

  This behavior is similar to that of InnoDB, where shorter primary keys save space not only in the primary index but in secondary indexes as well.

- If all columns that you access in a BDB table are part of the same index or part of the primary key, MySQL can execute the query without having to access the actual row. In a MyISAM table, this can be done only if the columns are part of the same index.

- Sequential scanning is slower for BDB tables than for MyISAM tables because the data in BDB tables is stored in B-trees and not in a separate data file.

- Key values are not prefix- or suffix-compressed like key values in MyISAM tables. In other words, key information takes a little more space in BDB tables compared to MyISAM tables.

- There are often holes in the BDB table to permit you to insert new rows in the middle of the index tree. This makes BDB tables somewhat larger than MyISAM tables.

- SELECT COUNT(*) FROM *tbl_name* is slow for BDB tables, because no row count is maintained in the table.

- The optimizer needs to know the approximate number of rows in the table. MySQL solves this by counting inserts and maintaining this in a separate segment in each BDB table. If you don't issue a lot of DELETE or ROLLBACK statements, this number should be accurate enough for the MySQL optimizer. However, MySQL stores the number only on close, so it may be incorrect if the server terminates unexpectedly. It should not be fatal even if this number is not 100% correct. You can update the row count by using ANALYZE TABLE or OPTIMIZE TABLE. See Section 12.4.2.1, “ANALYZE TABLE Syntax”, and Section 12.4.2.5, “OPTIMIZE TABLE Syntax”.

- Internal locking in BDB tables is done at the page level.

- LOCK TABLES works on BDB tables as with other tables. If you do not use LOCK TABLES, MySQL issues an internal multiple-write lock on the table (a lock that does not block other writers) to ensure that the table is properly locked if another thread issues a table lock.

- To support transaction rollback, the BDB storage engine maintains log files. For maximum performance, you can use the --bdb-logdir [1138] option to place the BDB logs on a different disk than the one where your databases are located.

- MySQL performs a checkpoint each time a new BDB log file is started, and removes any BDB log files that are not needed for current transactions. You can also use FLUSH LOGS at any time to checkpoint the Berkeley DB tables.

  For disaster recovery, you should use table backups plus MySQL's binary log. See Section 6.2, “Database Backup Methods”.

Warning

If you delete old log files that are still in use, BDB is not able to do recovery at all and you may lose data if something goes wrong.

- Applications must always be prepared to handle cases where any change of a BDB table may cause an automatic rollback and any read may fail with a deadlock error.
- If you get a full disk with a BDB table, you get an error (probably error 28) and the transaction should roll back. This contrasts with MyISAM and ISAM tables, for which mysqld waits for sufficient free disk space before continuing.

13.5.5 Restrictions on BDB Tables

The following list indicates restrictions that you must observe when using BDB tables:

- Each BDB table stores in its .db file the path to the file as it was created. This is done to enable detection of locks in a multi-user environment that supports symlinks. As a consequence of this, it is not possible to move BDB table files from one database directory to another.
- When making backups of BDB tables, you must either use mysqldump or else make a backup that includes the files for each BDB table (the .frm and .db files) as well as the BDB log files. The BDB storage engine stores unfinished transactions in its log files and requires them to be present when mysqld starts. The BDB logs are the files in the data directory with names of the form log.*NNNNNNNNN* (ten digits).
- If a column that permits NULL values has a unique index, only a single NULL value is permitted. This differs from other storage engines, which permit multiple NULL values in unique indexes.

13.5.6 Errors That May Occur When Using BDB Tables

- If the following error occurs when you start mysqld after upgrading, it means that the current version of BDB doesn't support the old log file format:

```
bdb:  Ignoring log file: .../log.NNNNNNNNN:
unsupported log version #
```

  In this case, you must delete all BDB logs from your data directory (the files that have names of the form log.*NNNNNNNNN*) and restart mysqld. We also recommend that you then use mysqldump --opt to dump your BDB tables, drop the tables, and restore them from the dump file.

- If autocommit mode is disabled and you drop a BDB table that is referenced in another transaction, you may get error messages of the following form in your MySQL error log:

```
001119 23:43:56  bdb:  Missing log fileid entry
001119 23:43:56  bdb:  txn_abort: Log undo failed for LSN:
                      1 3644744: Invalid
```

  This is not fatal, but the fix is not trivial. Avoid dropping BDB tables except while autocommit mode is enabled.

13.6 The EXAMPLE Storage Engine

The EXAMPLE storage engine was added in MySQL 4.1.3. It is a “stub” engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

The EXAMPLE storage engine is included in MySQL-Max binary distributions. To enable this storage engine if you build MySQL from source, invoke configure with the --with-example-storage-engine option.

To examine the source for the EXAMPLE engine, look in the sql/examples directory of a MySQL source distribution.

When you create an EXAMPLE table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. No other files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

The EXAMPLE storage engine does not support indexing.

13.7 The ARCHIVE Storage Engine

The ARCHIVE storage engine was added in MySQL 4.1.3. It is used for storing large amounts of data without indexes in a very small footprint.

The ARCHIVE storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke configure with the --with-archive-storage-engine option.

To examine the source for the ARCHIVE engine, look in the sql directory of a MySQL source distribution.

You can check whether the ARCHIVE storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_archive';
```

When you create an ARCHIVE table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. The storage engine creates other files, all having names beginning with the table name. The data and metadata files have extensions of .ARZ and .ARM, respectively. An .ARN file may appear during optimization operations.

The ARCHIVE engine supports INSERT and SELECT, but not DELETE, REPLACE, or UPDATE. It does support ORDER BY operations, BLOB columns, and basically all but spatial data types (see Section 16.4.1, “MySQL Spatial Data Types”). The ARCHIVE engine uses row-level locking.

Storage: Rows are compressed as they are inserted. The ARCHIVE engine uses zlib lossless data compression (see http://www.zlib.net/). You can use OPTIMIZE TABLE to analyze the table and pack it into a smaller format (for a reason to use OPTIMIZE TABLE, see later in this section). There are several types of insertions that are used:

- An INSERT statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A SELECT forces a flush to occur, unless the only insertions that have come in were INSERT DELAYED (those flush as necessary). See Section 12.2.4.2, “INSERT DELAYED Syntax”.
- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A SELECT never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A SELECT operation performs a complete table scan: When a SELECT occurs, it finds out how many rows are currently

available and reads that number of rows. SELECT is performed as a consistent read. Note that lots of SELECT statements during insertion can deteriorate the compression, unless only bulk or delayed inserts are used. To achieve better compression, you can use OPTIMIZE TABLE or REPAIR TABLE. The number of rows in ARCHIVE tables reported by SHOW TABLE STATUS is always accurate. See Section 12.4.2.5, “OPTIMIZE TABLE Syntax”, Section 12.4.2.6, “REPAIR TABLE Syntax”, and Section 12.4.5.23, “SHOW TABLE STATUS Syntax”.

Additional Resources

- A forum dedicated to the ARCHIVE storage engine is available at http://forums.mysql.com/list.php?112.

13.8 The CSV Storage Engine

The CSV storage engine was added in MySQL 4.1.4. This engine stores data in text files using comma-separated values format. It is unavailable on Windows until MySQL 5.1.

The CSV storage engine is included in MySQL-Max binary distributions. To enable this storage engine if you build MySQL from source, invoke configure with the --with-csv-storage-engine option.

To examine the source for the CSV engine, look in the sql/examples directory of a MySQL source distribution.

When you create a CSV table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. The storage engine also creates a data file. Its name begins with the table name and has a .CSV extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)
    -> ENGINE = CSV;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
+------+------------+
| i    | c          |
+------+------------+
|    1 | record one |
|    2 | record two |
+------+------------+
2 rows in set (0.00 sec)
```

If you examine the test.CSV file in the database directory created by executing the preceding statements, its contents should look like this:

```
"1","record one"
"2","record two"
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

The CSV storage engine does not support indexing.

13.9 The BLACKHOLE Storage Engine

The BLACKHOLE storage engine was added in MySQL 4.1.11. This engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

The BLACKHOLE storage engine is included in MySQL-Max binary distributions. To enable this storage engine if you build MySQL from source, invoke `configure` with the `--with-blackhole-storage-engine` option.

To examine the source for the BLACKHOLE engine, look in the `sql` directory of a MySQL source distribution.

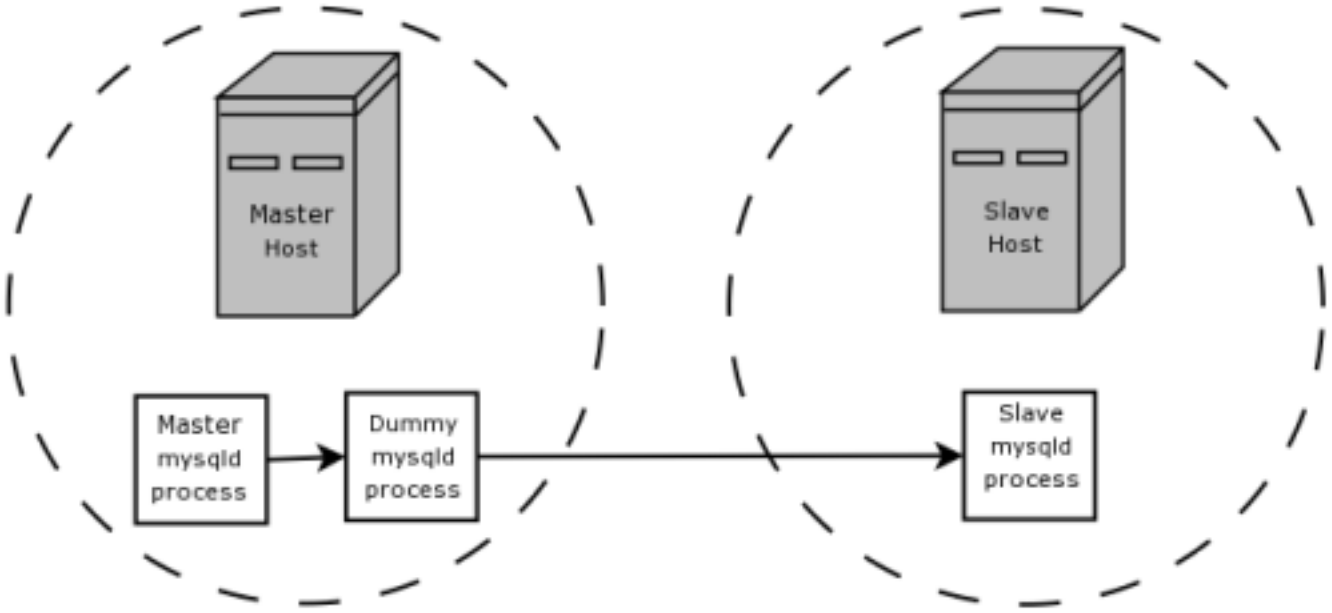
When you create a BLACKHOLE table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. There are no other files associated with the table.

The BLACKHOLE storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

You can check whether the BLACKHOLE storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_blackhole_engine';
```

Inserts into a BLACKHOLE table do not store any data, but if the binary log is enabled, the SQL statements are logged (and replicated to slave servers). This can be useful as a repeater or filter mechanism. Suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is BLACKHOLE, depicted as follows:



The master writes to its binary log. The “dummy” `mysqld` process acts as a slave, applying the desired combination of `replicate-do-*` and `replicate-ignore-*` rules, and writes a new, filtered binary log of its own. (See Section 14.8, “Replication and Binary Logging Options and Variables”.) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional `mysqld` process on the replication master host. This type of setup can be repeated with additional replication slaves.

Other possible uses for the `BLACKHOLE` storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using `BLACKHOLE` with and without binary logging enabled.
- `BLACKHOLE` is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

13.10 The `ISAM` Storage Engine

The original storage engine in MySQL was the `ISAM` engine. It was the only storage engine available until MySQL 3.23, when the improved `MyISAM` engine was introduced as the default. `ISAM` is deprecated. As of MySQL 4.1, it is included in the source but not enabled in binary distributions. It is not available in MySQL 5.0. Embedded MySQL server versions do not support `ISAM` tables by default.

Due to the deprecated status of `ISAM`, and because `MyISAM` is an improvement over `ISAM`, you are advised to convert any remaining `ISAM` tables to `MyISAM` as soon as possible. To convert an `ISAM` table to a `MyISAM` table, use an `ALTER TABLE` statement:

```
mysql> ALTER TABLE tbl_name TYPE = MYISAM;
```

For more information about `MyISAM`, see Section 13.1, “The `MyISAM` Storage Engine”.

Each `ISAM` table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table definition. The data file has an `.ISD` extension. The index file has an `.ISM` extension.

`ISAM` uses B-tree indexes.

You can check or repair `ISAM` tables with the `isamchk` utility. See Section 6.6.1, “Using `myisamchk` for Crash Recovery”.

`ISAM` has the following properties:

- Compressed and fixed-length keys
- Fixed and dynamic record length
- 16 indexes per table, with 16 key parts per key
- Maximum key length 256 bytes (default)
- Data values are stored in machine format; this is fast, but machine/OS dependent

Many of the properties of `MyISAM` tables are also true for `ISAM` tables. However, there are also many differences. The following list describes some of the ways that `ISAM` is distinct from `MyISAM`:

- Not binary portable across OS/platforms.
- Can't handle tables larger than 4GB.
- Only supports prefix compression on strings.
- Smaller (more restrictive) key limits.
- Dynamic tables become more fragmented.
- Doesn't support `MERGE` tables.
- Tables are checked and repaired with `isamchk` rather than with `myisamchk`.
- Tables are compressed with `pack_isam` rather than with `myisampack`.
- Cannot be used with the `BACKUP TABLE` or `RESTORE TABLE` backup-related statements.
- Cannot be used with the `CHECK TABLE`, `REPAIR TABLE`, `OPTIMIZE TABLE`, or `ANALYZE TABLE` table-maintenance statements.
- No support for full-text searching or spatial data types.
- No support for multiple character sets per table.
- Indexes cannot be assigned to specific key caches.

Chapter 14 Replication

Table of Contents

Replication capabilities enabling the databases on one MySQL server to be duplicated on another were introduced in MySQL 3.23.15. This chapter describes the various replication features provided by MySQL.

It introduces replication concepts, shows how to set up replication servers, and serves as a reference to the available replication options. It also provides a list of frequently asked questions (with answers), and troubleshooting advice for solving problems.

For a description of the syntax of replication-related SQL statements, see Section 12.5, “Replication Statements”.

14.1 Introduction to Replication

MySQL 3.23.15 and up features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of MySQL Cluster (see Chapter 15, *MySQL Cluster*).

In single-master replication, the master server writes updates to its binary log files and maintains an index of those files to keep track of log rotation. The binary log files serve as a record of updates to be sent to any slave servers. When a slave connects to its master, it informs the master of the position up to which the slave read the logs at its last successful update. The slave receives any updates that have taken place since that time, and then blocks and waits for the master to notify it of new updates.

A slave server can itself serve as a master if you want to set up chained replication servers.

When you are using replication, all updates to the tables that are replicated should be performed on the master server. Otherwise, you must always be careful to avoid conflicts between updates that users make to tables on the master and updates that they make to tables on the slave.

Replication offers benefits for robustness, speed, and system administration:

- Robustness is increased with a master/slave setup. In the event of problems with the master, you can switch to the slave as a backup.
- Better response time for clients can be achieved by splitting the load for processing client queries between the master and slave servers. `SELECT` queries may be sent to the slave to reduce the query processing load of the master. Statements that modify data should still be sent to the master so that the master and slave do not get out of synchrony. This load-balancing strategy is effective if nonupdating queries dominate, but that is the normal case.
- Another benefit of using replication is that you can perform database backups using a slave server without disturbing the master. The master continues to process updates while the backup is being made. See Section 6.2, “Database Backup Methods”.

14.2 Replication Implementation Overview

MySQL replication is based on the master server keeping track of all changes to your databases (updates, deletes, and so on) in its binary logs. Therefore, to use replication, you must enable binary logging on the master server. See Section 5.3.4, “The Binary Log”.

Each slave server receives from the master the saved updates that the master has recorded in its binary log, so that the slave can execute the same updates on its copy of the data.

It is *extremely* important to realize that the binary log is simply a record starting from the fixed point in time at which you enable binary logging. Any slaves that you set up need copies of the databases on your master *as they existed at the moment you enabled binary logging on the master*. If you start your slaves with databases that are not in the same state as those on the master when the binary log was started, your slaves are quite likely to fail.

After the slave has been set up with a copy of the master's data, it connects to the master and waits for updates to process. If the master fails, or the slave loses connectivity with your master, the slave keeps

trying to connect periodically until it is able to resume listening for updates. The `CHANGE MASTER TO` statement or `--master-connect-retry` [1173] option controls the retry interval. The default is 60 seconds.

Each slave keeps track of where it left off when it last read from its master server. The master has no knowledge of how many slaves it has or which ones are up to date at any given time.

14.3 Replication Implementation Details

MySQL replication capabilities are implemented using three threads, one on the master server and two on the slave:

- **Binlog dump thread.** The master creates a thread to send the binary log contents to a slave when the slave connects. This thread can be identified in the output of `SHOW PROCESSLIST` on the master as the `Binlog Dump` thread.

  The binlog dump thread acquires a lock on the master's binary log for reading each event that is to be sent to the slave. As soon as the event has been read, the lock is released, even before the event is sent to the slave.

- **Slave I/O thread.** When a `START SLAVE` statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs.

  The slave I/O thread reads the updates that the master's `Binlog Dump` thread sends (see previous item) and copies them to local files that comprise the slave's relay log.

  The state of this thread is shown as `Slave_IO_running` in the output of `SHOW SLAVE STATUS` or as `Slave_running` in the output of `SHOW STATUS`.

- **Slave SQL thread.** The slave creates an SQL thread to read the relay log that is written by the slave I/O thread and execute the events contained therein.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one binlog dump thread for each currently connected slave, and each slave has its own I/O and SQL threads.

Note

For versions of MySQL before 4.0.2, replication involves only two threads (one on the master and one on the slave). The slave I/O and SQL threads are combined as a single thread, and no relay log files are used.

A slave uses two threads to separate reading updates from the master and executing them into independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts. This enables the master server to purge its binary logs sooner because it no longer needs to wait for the slave to fetch their contents.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication. For information on master states, see Section 7.11.4, “Replication Master Thread States”. For slave states, see Section 7.11.5, “Replication Slave I/O Thread States”, and Section 7.11.6, “Replication Slave SQL Thread States”.

The following example illustrates how the three threads show up in the output from `SHOW PROCESSLIST`. The output format is that used by `SHOW PROCESSLIST` as of MySQL 4.0.15, when the content of the `State` column was changed to be more meaningful compared to earlier versions.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
   Info: NULL
```

Here, thread 2 is a `Binlog Dump` replication thread that services a connected slave. The `State` information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no `Binlog Dump` threads on a master server, this means that replication is not running; that is, no slaves are currently connected.

On a slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
   Info: NULL
```

The `State` information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See Section 14.10, “Replication FAQ”. If sufficient time elapses on the master side without activity on the `Binlog Dump` thread, the master determines that the slave is no longer connected. As for any other client connection, the timeouts for this depend on the values of `net_write_timeout` and `net_retry_count`; for more information about these, see Section 14.8, “Replication and Binary Logging Options and Variables”.

14.3.1 Replication Relay and Status Files

During replication, a slave server creates several files that hold the binary log events relayed from the master to the slave, and to record information about the current status and location within the relay log. There are three file types used in the process:

- The *relay log* consists of the events read from the binary log of the master and written by the slave I/O thread. Events in the relay log are executed on the slave as part of the SQL thread.
- The *master.info* file contains the status and current configuration information for the slave's connectivity to the master. The file holds information on the master host name, login credentials, and coordinates indicating how far the slave has read from the master's binary log.
- The *relay-log.info* file holds the status information about the execution point within the slave's relay log.

14.3.2 The Slave Relay Log

The relay log, like the binary log, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

The term “relay log file” generally denotes an individual numbered file containing database events. The term “relay log” collectively denotes the set of numbered relay log files plus the index file.

Relay log files have the same format as binary log files and can be read using `mysqlbinlog` (see Section 4.6.6, “`mysqlbinlog` — Utility for Processing Binary Log Files”).

By default, relay log file names have the form *`host_name`*`-relay-bin.`*`nnnnnn`* in the data directory, where *`host_name`* is the name of the slave server host and *`nnnnnn`* is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001` (`001` in MySQL 4.0 or older). The slave uses an index file to track the relay log files currently in use. The default relay log index file name is *`host_name`*`-relay-bin.index` in the data directory.

The default relay log file and relay log index file names can be overridden with, respectively, the `--relay-log` [1175] and `--relay-log-index` [1175] server options (see Section 14.8, “Replication and Binary Logging Options and Variables”).

If a slave uses the default host-based relay log file names, changing a slave's host name after replication has been set up can cause replication to fail with the errors `Failed to open the relay log` and `Could not find target log during relay log initialization`. This is a known issue (see Bug #2122). If you anticipate that a slave's host name might change in the future (for example, if networking is set up on the slave such that its host name can be modified using DHCP), you can avoid this issue entirely by using the `--relay-log` [1175] and `--relay-log-index` [1175] options to specify relay log file names explicitly when you initially set up the slave. This will make the names independent of server host name changes.

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed; for example, with `FLUSH LOGS` or `mysqladmin flush-logs`. (This creates a new relay log only as of MySQL 4.0.14.)
- When the size of the current relay log file becomes “too large,” determined as follows:
  - If the value of `max_relay_log_size` [420] is greater than 0, that is the maximum relay log file size.
  - If the value of `max_relay_log_size` [420] is 0, `max_binlog_size` [1184] determines the maximum relay log file size. `max_binlog_size` [1184] always determines the relay log size before MySQL 4.0.14, the first version in which `max_relay_log_size` [420] appears.

The SQL thread automatically deletes each relay log file as soon as it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread

takes care of doing so. However, as of MySQL 4.0.14, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

14.3.3 The Slave Status Files

A slave replication server creates two small status files. By default, these files are named `master.info` and `relay-log.info` and created in the data directory. Their names and locations can be changed by using the `--master-info-file` [1174] and `--relay-log-info-file` [1175] options. See Section 14.8, "Replication and Binary Logging Options and Variables".

The two status files contain information like that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in Section 12.5.2, "SQL Statements for Controlling Slave Servers". Because the status files are stored on disk, they survive a slave server's shutdown. The next time the slave starts up, it reads the two files to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The `master.info` file should be protected because it contains the password for connecting to the master. See Section 5.4.2.1, "Administrator Guidelines for Password Security".

The slave I/O thread updates the `master.info` file. As of MySQL 4.1, the file includes a line count and information about SSL options. The following table shows the correspondence between the lines in the file and the columns displayed by `SHOW SLAVE STATUS`.

Line	Description
1	Number of lines in the file
2	`Master_Log_File`
3	`Read_Master_Log_Pos`
4	`Master_Host`
5	`Master_User`
6	Password (not shown by `SHOW SLAVE STATUS`)
7	`Master_Port`
8	`Connect_Retry`
9	`Master_SSL_Allowed`
10	`Master_SSL_CA_File`
11	`Master_SSL_CA_Path`
12	`Master_SSL_Cert`
13	`Master_SSL_Cipher`
14	`Master_SSL_Key`

Before MySQL 4.1, the file does not include a line count or information about SSL options.

Line	Description
1	`Master_Log_File`
2	`Read_Master_Log_Pos`
3	`Master_Host`
4	`Master_User`
5	Password (not shown by `SHOW SLAVE STATUS`)
6	`Master_Port`

Line	Description
7	`Connect_Retry`

The slave SQL thread updates the `relay-log.info` file. The following table shows the correspondence between the lines in the file and the columns displayed by `SHOW SLAVE STATUS`.

Line	Description
1	`Relay_Log_File`
2	`Relay_Log_Pos`
3	`Relay_Master_Log_File`
4	`Exec_Master_Log_Pos`

The contents of the `relay-log.info` file and the states shown by the `SHOW SLAVE STATUS` statement might not match if the `relay-log.info` file has not been flushed to disk. Ideally, you should only view `relay-log.info` on a slave that is offline (that is, `mysqld` is not running). For a running system, `SHOW SLAVE STATUS` should be used.

When you back up the slave's data, you should back up these two status files as well, along with the relay log files. They are needed to resume replication after you restore the slave's data. If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. This requires that the binary logs still exist on the master server.

14.4 How to Set Up Replication

This section briefly describes how to set up complete replication of a MySQL server. It assumes that you want to replicate all databases on the master and have not previously configured replication. You must shut down your master server briefly to complete the steps outlined here.

This procedure is written in terms of setting up a single slave, but you can repeat it to set up multiple slaves.

Although this method is the most straightforward way to set up a slave, it is not the only one. For example, if you have a snapshot of the master's data, and the master already has its server ID set and binary logging enabled, you can set up a slave without shutting down the master or even blocking updates to it. For more details, please see Section 14.10, “Replication FAQ”.

If you want to administer a MySQL replication setup, we suggest that you read this entire chapter through and try all statements mentioned in Section 12.5.1, “SQL Statements for Controlling Master Servers”, and Section 12.5.2, “SQL Statements for Controlling Slave Servers”. You should also familiarize yourself with the replication startup options described in Section 14.8, “Replication and Binary Logging Options and Variables”.

Note

This procedure and some of the replication SQL statements shown in later sections refer to the `SUPER` [493] privilege. Prior to MySQL 4.0.2, use the `PROCESS` [492] privilege instead.

1. Make sure that you have a recent version of MySQL installed on the master and slaves, and that these versions are compatible according to the table shown in Section 14.5, “Replication Compatibility Between MySQL Versions”.

If you encounter a problem, please do not report it as a bug until you have verified that the problem is present in the latest MySQL release.

2. Set up an account on the master server that the slave server can use to connect. This account must be given the `REPLICATION SLAVE` [492] privilege. If the account is used only for replication (which is recommended), you need not grant any additional privileges.

   Suppose that your domain is `mydomain.com` and that you want to create an account with a user name of `repl` such that slave servers can use the account to access the master server from any host in your domain using a password of `slavepass`. To create the account, use this `GRANT` statement:

   ```
   mysql> GRANT REPLICATION SLAVE ON *.*
       -> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
   ```

   For MySQL versions older than 4.0.2, the `REPLICATION SLAVE` [492] privilege does not exist. Grant the `FILE` [491] privilege instead:

   ```
   mysql> GRANT FILE ON *.*
       -> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
   ```

   For additional information about setting up user accounts and privileges, see Section 5.6, “MySQL User Account Management”.

3. Flush all the tables and block write statements by executing a `FLUSH TABLES WITH READ LOCK` statement:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   For example, if you are using `InnoDB` tables, you should use the `InnoDB Hot Backup` tool to obtain a consistent snapshot. This tool records the log name and offset corresponding to the snapshot to be later used on the slave. `Hot Backup` is a nonfree (commercial) tool that is not included in the standard MySQL distribution. See the `InnoDB Hot Backup` home page at http://www.innodb.com/wp/products/hot-backup/ for detailed information.

   Otherwise, you can obtain a reliable binary snapshot of `InnoDB` tables only after shutting down the MySQL Server.

   An alternative that works for both `MyISAM` and `InnoDB` tables is to take an SQL dump of the master instead of a binary copy as described in the preceding discussion. For this, you can use `mysqldump --master-data` on your master and later load the SQL dump file into your slave. However, this is slower than doing a binary copy.

   Leave running the client from which you issue the `FLUSH TABLES` statement so that the read lock remains in effect. (If you exit the client, the lock is released.) Then take a snapshot of the data on your master server.

   The easiest way to create a snapshot is to use an archiving program to make a binary backup of the databases in your master's data directory. For example, use `tar` on Unix, or `PowerArchiver`, `WinRAR`, `WinZip`, or any similar software on Windows. To use `tar` to create an archive that includes all databases, change location into the master server's data directory, then execute this command:

   ```
   shell> tar -cvf /tmp/mysql-snapshot.tar .
   ```

   If you want the archive to include only a database called `this_db`, use this command instead:

```
shell> tar -cvf /tmp/mysql-snapshot.tar ./this_db
```

Then copy the archive file to the `/tmp` directory on the slave server host. On that machine, change location into the slave's data directory, and unpack the archive file using this command:

```
shell> tar -xvf /tmp/mysql-snapshot.tar
```

You may not want to replicate the `mysql` database if the slave server has a different set of user accounts from those that exist on the master. In this case, you should exclude it from the archive. You also need not include any log files in the archive, or the `master.info` or `relay-log.info` files.

While the read lock placed by `FLUSH TABLES WITH READ LOCK` is in effect, read the value of the current binary log name and offset on the master:

```
mysql > SHOW MASTER STATUS;
+--------------+----------+--------------+------------------+
| File         | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
```

The `File` column shows the name of the log and `Position` shows the offset within the file. In this example, the binary log file is `mysql-bin.003` and the offset is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

If the master has been running previously without binary logging enabled, the log name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (`''`) and `4`.

After you have taken the snapshot and recorded the log name and offset, you can re-enable write activity on the master:

```
mysql> UNLOCK TABLES;
```

4. Make sure that the `[mysqld]` section of the `my.cnf` file on the master host includes a `log-bin` option. The section should also have a `server-id=`*`master_id`* option, where *`master_id`* must be a positive integer value from 1 to 2^{32} – 1. For example:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

If those options are not present, add them and restart the server. The server cannot act as a replication master unless binary logging is enabled.

Note

For the greatest possible durability and consistency in a replication setup using `InnoDB` with transactions, you should use `innodb_flush_log_at_trx_commit=1`, `sync_binlog=1`, and `innodb-safe-binlog` in your master `my.cnf` file.

5. Stop the server that is to be used as a slave and add the following lines to its `my.cnf` file:

```
[mysqld]
server-id=slave_id
```

   The *`slave_id`* value, like the *`master_id`* value, must be a positive integer value from 1 to 2^{32} – 1. In addition, it is necessary that the ID of the slave be different from the ID of the master. For example:

```
[mysqld]
server-id=2
```

   If you are setting up multiple slaves, each one must have a unique `server-id` [1167] value that differs from that of the master and from each of the other slaves. Think of `server-id` values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

   If you do not specify a `server-id` [1167] value, it defaults to 0.

   Note

   If you omit `server-id` [1167] (or set it explicitly to 0), a master refuses connections from all slaves, and a slave refuses to connect to a master. Thus, omitting `server-id` [1167] is good only for backup with a binary log.

6. If you made a binary backup of the master server's data, copy it to the slave server's data directory before starting the slave. Make sure that the privileges on the files and directories are correct. The system account that you use to run the slave server must be able to read and write the files, just as on the master.

   If you made a backup using `mysqldump`, start the slave first. The dump file is loaded in a later step.

7. Start the slave server. If it has been replicating previously, start the slave server with the `--skip-slave-start` [1179] option so that it does not immediately try to connect to its master. You also may want to start the slave server with the `--log-warnings` [389] option to get more messages in the error log about problems (for example, network or connection problems). The option is enabled by default as of MySQL 4.0.19 and 4.1.2, but as of MySQL 4.0.21 and 4.1.3, aborted connections are not logged to the error log unless the value is greater than 1.

8. If you made a backup of the master server's data using `mysqldump`, load the dump file into the slave server:

```
shell> mysql -u root -p < dump_file.sql
```

9. Execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='replication_user_name',
    ->     MASTER_PASSWORD='replication_password',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

The following table shows the maximum permissible length for the string-valued options.

Option	Maximum Length
`MASTER_HOST`	60
`MASTER_USER`	16
`MASTER_PASSWORD`	32
`MASTER_LOG_FILE`	255

10. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` option for the master, slaves cannot connect to it.

If you have forgotten to set the `server-id` option for the slave, you get the following error in the slave's error log:

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

Once a slave is replicating, you can find in its data directory one file named `master.info` and another named `relay-log.info`. The slave uses these two files to keep track of how much of the master's binary log it has processed. Do *not* remove or edit these files unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The slave will use the values specified in the statement to update the status files automatically.

Note

The content of `master.info` overrides some of the server options specified on the command line or in `my.cnf`. See Section 14.8, “Replication and Binary Logging Options and Variables”, for more details.

Once you have a snapshot of the master, you can use it to set up other slaves by following the slave portion of the procedure just described. You do not need to take another snapshot of the master; you can use the same one for each slave.

14.5 Replication Compatibility Between MySQL Versions

MySQL supports replication from one major version to the next higher major version. For example, you can replicate from a master running MySQL 4.0 to a slave running MySQL 4.1, from a master running MySQL 4.1 to a slave running MySQL 5.0, and so on.

Note

The original binary log format was developed in MySQL 3.23. It was changed in MySQL 4.0.

You *cannot* replicate from a master that uses a newer binary log format to a slave that uses an older format—for example, from MySQL 4.1 to MySQL 3.23. (In general, MySQL does not support replication from newer masters to older slaves.) This also has significant consequences for upgrading servers in a replication setup, as described in Section 14.6, “Upgrading a Replication Setup”.

As far as replication is concerned, the binary log format used by all MySQL 4.0 and MySQL 4.1 releases is identical. However, replication from a 4.1 master to a 4.0 slave is unsupported; it has not been tested thoroughly, and no further development or bug fixing is planned for this master/slave combination. Although the binary log format is the same for 4.0 and 4.1, there are other constraints, such as SQL-level compatibility issues. For example, a 4.1 master cannot replicate to a 4.0 slave if the replicated statements use SQL features available in 4.1 but not 4.0.

In some cases, it is also possible to replicate between a master and a slave that is more than one major version newer than the master. However, there are known issues with trying to replicate from a master running MySQL 4.1 or earlier to a slave running MySQL 5.1 or later. To work around such problems, you can insert a MySQL server running an intermediate version between the two; for example, rather than replicating directly from a MySQL 4.1 master to a MySQL 5.1 slave, it is possible to replicate from a MySQL 4.1 server to a MySQL 5.0 server, and then from the MySQL 5.0 server to a MySQL 5.1 server.

Important

It is strongly recommended to use the most recent release available within a given MySQL major version because replication (and other) capabilities are continually being improved. It is also recommended to upgrade masters and slaves that use alpha or beta releases of a major version of MySQL to GA (production) releases when these become available for that major version.

For more information on potential replication issues, see Section 14.7, “Replication Features and Issues”.

14.6 Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

14.6.1 Upgrading Replication to 4.0 or 4.1

This section applies to upgrading replication from MySQL 3.23 to 4.0 or 4.1. A 4.0 server should be 4.0.3 or newer, as mentioned in Section 14.5, “Replication Compatibility Between MySQL Versions”.

When you upgrade a master from MySQL 3.23 to MySQL 4.0 or 4.1, you should first ensure that all the slaves of this master are at 4.0 or 4.1. If that is not the case, you should first upgrade your slaves: Shut down each one, upgrade it, restart it, and restart replication.

The upgrade can safely be done using the following procedure, assuming that you have a 3.23 master to upgrade and the slaves are 4.0 or 4.1. Note that after the master has been upgraded, you should not restart replication using any old 3.23 binary logs, because this unfortunately confuses the 4.0 or 4.1 slaves.

1. Block all updates on the master by issuing a `FLUSH TABLES WITH READ LOCK` statement.

2. Wait until all the slaves have caught up with all changes from the master server. Use `SHOW MASTER STATUS` on the master to obtain its current binary log file and position. Then, for each slave, use those values with a `SELECT MASTER_POS_WAIT()` statement. The statement blocks on the slave and returns when the slave has caught up. Then run `STOP SLAVE` on the slave.

3. Stop the master server and upgrade it to MySQL 4.0 or 4.1.

4. Restart the master server and record the name of its newly created binary log. You can obtain the name of the file by issuing a `SHOW MASTER STATUS` statement on the master. Then issue these statements on each slave:

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE='binary_log_name',
    ->     MASTER_LOG_POS=4;
mysql> START SLAVE;
```

14.7 Replication Features and Issues

The following sections provide information about what is supported and what is not in MySQL replication, and about specific issues and situations that may occur when replicating certain statements.

In general, replication compatibility at the SQL level requires that any features used be supported by both the master and the slave servers. If you use a feature on a master server that is available only as of a given version of MySQL, you cannot replicate to a slave that is older than that version. Such incompatibilities are likely to occur between series, so that, for example, you cannot replicate from MySQL 4.1 to 4.0. However, these incompatibilities also can occur for within-series replication. For example, the `CONVERT_TZ()` [828] function is available in MySQL 4.1.3 and up. If you use this function on the master server, you cannot replicate to a slave server that is older than MySQL 4.1.3.

Additional information specific to `InnoDB` and replication is given in Section 13.2.5.5, “`InnoDB` and MySQL Replication”.

14.7.1 Replication and `AUTO_INCREMENT`

Replication of `AUTO_INCREMENT`, `LAST_INSERT_ID()` [874], and `TIMESTAMP` values is done correctly, subject to the following exceptions.

- `INSERT DELAYED ... VALUES(LAST_INSERT_ID())` inserts a different value on the master and the slave. (Bug #20819)

- Adding an `AUTO_INCREMENT` column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to the table `t1` that has columns `col1` and `col2`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both master and slave, *all* columns of `t1` must be referenced in the `ORDER BY` clause.

The instructions just given are subject to the limitations of `CREATE TABLE ... LIKE`: Foreign key definitions are ignored, as are the `DATA DIRECTORY` and `INDEX DIRECTORY` table options. If a table definition includes any of those characteristics, create `t2` using a `CREATE TABLE` statement that is identical to the one used to create `t1`, but with the addition of the `AUTO_INCREMENT` column.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also Section B.5.7.1, “Problems with `ALTER TABLE`”.

14.7.2 Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

- You must *always* use the same *global* character set and collation on the master and the slave. (These are controlled by the `--character-set-server` [385] and `--collation-server` [385] options.) Otherwise, you may get duplicate-key errors on the slave, because a key that is unique in the master character set might not be unique in the slave character set.
- If the master is older than MySQL 4.1.3, the character set of any client should never be made different from its global value because this character set change is not known to the slave. In other words, clients should not use `SET NAMES`, `SET CHARACTER SET`, and so forth. If both the master and the slave are 4.1.3 or newer, clients can freely set session values for character set variables because these settings are written to the binary log and so are known to the slave. That is, clients can use `SET NAMES` or `SET CHARACTER SET` or can set variables such as `collation_client` or `collation_server` [409]. However, clients are prevented from changing the *global* value of these variables; as stated previously, the master and slave must always have identical global character set values.
- If the master has databases with a character set different from the global `character_set_server` [408] value, you should design your `CREATE TABLE` statements so that they do not implicitly rely on the database default character set, because there currently is a bug (Bug #2326). A good workaround is to state the character set and collation explicitly in `CREATE TABLE` statements.

14.7.3 Replication and `DIRECTORY` Table Options

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host file system or if it exists but is not accessible to the slave server. As of MySQL 4.0.15, there is an `sql_mode` [429] option called `NO_DIR_IN_CREATE` [459]. If the slave server is run with this SQL mode enabled, it ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

14.7.4 Replication and Floating-Point Values

Floating-point values are approximate, so comparisons involving them are inexact. This is true for operations that use floating-point values explicitly, or values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See Section 11.2, “Type Conversion in Expression Evaluation”, and Section B.5.5.8, “Problems with Floating-Point Values”.

14.7.5 Replication and FLUSH

Before MySQL 4.1.1, the FLUSH, ANALYZE TABLE, OPTIMIZE TABLE, and REPAIR TABLE statements are not written to the binary log and thus are not replicated to the slaves. This is not normally a problem because these statements do not modify table data.

However, this behavior can cause difficulties under certain circumstances. If you replicate the privilege tables in the mysql database and update those tables directly without using the GRANT statement, you must issue a FLUSH PRIVILEGES statement on your slaves to put the new privileges into effect. Also if you use FLUSH TABLES when renaming a MyISAM table that is part of a MERGE table, you must issue FLUSH TABLES manually on the slaves.

As of MySQL 4.1.1, these statements are written to the binary log (unless you specify NO_WRITE_TO_BINLOG or its alias LOCAL). Exceptions are that FLUSH LOGS, FLUSH MASTER, FLUSH SLAVE, and FLUSH TABLES WITH READ LOCK are not logged in any case because they may cause problems if replicated to a slave. For a syntax example, see Section 12.4.6.2, “FLUSH Syntax”.

14.7.6 Replication and System Functions

Certain functions do not replicate well under some conditions:

- The USER() [876], CURRENT_USER() [872], UUID() [879], and LOAD_FILE() [798] functions are replicated without change and thus do not work reliably on the slave. This is also true for CONNECTION_ID() [872] in slave versions older than 4.1.1. The new implementation of the PASSWORD() [868] function in MySQL 4.1 is well replicated in masters from 4.1.1 and up, but your slaves also must be 4.1.1 or above to replicate it. If you have older slaves and need to replicate PASSWORD() [868] from your 4.1.x master, you must start your master with the --old-passwords [391] option, so that it uses the old implementation of PASSWORD() [868].

  The PASSWORD() [868] implementation in MySQL 4.1.0 differs from every other version of MySQL. Avoid using 4.1.0 in a replication scenario.

- The GET_LOCK() [877], RELEASE_LOCK() [879], IS_FREE_LOCK() [879], and IS_USED_LOCK() [879] functions that handle user-level locks are replicated without the slave knowing the concurrency context on the master. Therefore, these functions should not be used to insert into a master table because the content on the slave would differ. For example, do not issue a statement such as INSERT INTO mytable VALUES(GET_LOCK(...)).

- The FOUND_ROWS() [872] function is not replicated reliably. A workaround is to store the result of the function call in a user variable, and then use that in the INSERT statement. For example, if you wish to store the result in a table named mytable, you might normally do so like this:

  ```
  SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
  INSERT INTO mytable VALUES( FOUND_ROWS() );
  ```

  However, if you are replicating mytable, then you should use SELECT INTO, and then store the variable in the table, like this:

  ```
  SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
  INSERT INTO mytable VALUES(@found_rows);
  ```

  In this way, the user variable is replicated as part of the context, and applied on the slave correctly.

- In 3.23, RAND() [822] in updates does not replicate properly. Use RAND(*deterministic_expression*) [822] if you are replicating updates with RAND() [822]. You can, for example, use UNIX_TIMESTAMP() [841] as the argument to RAND() [822].

14.7.7 Replication and LIMIT

DELETE, UPDATE, and INSERT ... SELECT statements containing a LIMIT clause are not guaranteed to produce the same result on the slave as on the master, since the order of the rows affected is not defined. Such statements can be replicated correctly only if they also contain an ORDER BY clause.

14.7.8 Replication and LOAD Operations

If on the master a LOAD DATA INFILE is interrupted (for example, by a integrity constraint violation or a killed connection), the slave skips this LOAD DATA INFILE entirely. This means that if this statement permanently inserted or updated table records before being interrupted, these modifications are *not* replicated to the slave.

In addition, LOAD DATA INFILE does not replicate correctly when --binlog-do-db [1182] is used. (Bug #19662)

LOAD DATA INFILE also does not replicate well from 4.0 and earlier masters to 5.1 or later slaves. In such cases, it is best to upgrade the master to 5.0 or later. (Bug #31240)

The LOAD DATA INFILE statement CONCURRENT option is not replicated; that is, LOAD DATA CONCURRENT INFILE is replicated as LOAD DATA INFILE, and LOAD DATA CONCURRENT LOCAL INFILE is replicated as LOAD DATA LOCAL INFILE. (Bug #34628)

When a 4.x slave replicates a LOAD DATA INFILE from a 3.23 master, the values of the Exec_Master_Log_Pos and Relay_Log_Space columns of SHOW SLAVE STATUS become incorrect. The inaccuracy in Exec_Master_Log_Pos causes problems when you stop and restart replication, so it is a good idea to correct the value before this by executing FLUSH LOGS on the master.

The following problems with replication in MySQL 3.23 are fixed in MySQL 4.0:

- LOAD DATA INFILE is handled properly, as long as the data file still resides on the master server at the time of update propagation.
- LOAD DATA LOCAL INFILE is no longer skipped on the slave as it was in 3.23.

14.7.9 Replication and the Slow Query Log

Replication slaves do not write replicated queries to the slow query log, even if the same queries were written to the slow query log on the master. This is a known issue. (Bug #23300)

14.7.10 Replication and Master or Slave Shutdowns

It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the CHANGE MASTER TO statement or --master-connect-retry [1173] option. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for slave_net_timeout [1181] seconds. If your outages are short, you may want to decrease slave_net_timeout [1181]. See Section 5.1.3, “Server System Variables”.

An unclean shutdown (for example, a crash) on the master side can result in the master binary log having a final position less than the most recent position read by the slave, due to the master binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting sync_binlog=1 [1184] in the master my.cnf file helps to minimize this problem because it causes the master to flush its binary log more frequently.

Unclean master shutdowns may cause inconsistencies between the content of tables and the binary log. This can be avoided by using InnoDB tables and the --innodb-safe-binlog [387] option on the master. See Section 5.3.4, “The Binary Log”.

Shutting down a slave cleanly is safe because it keeps track of where it left off. However, be careful that the slave does not have temporary tables open; see Section 14.7.12, “Replication and Temporary Tables”. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the problem occurred:

- For transactions, the slave commits and then updates relay-log.info. If a crash occurs between these two operations, relay log processing will have proceeded further than the information file indicates and the slave will re-execute the events from the last transaction in the relay log after it has been restarted.
- A similar problem can occur if the slave updates relay-log.info but the server host crashes before the write has been flushed to disk. Writes are not forced to disk because the server relies on the operating system to flush the file from time to time.

The fault tolerance of your system for these types of problems is greatly increased if you have a good uninterruptible power supply.

14.7.11 Replication and MEMORY Tables

When a server shuts down and restarts, its MEMORY (HEAP) tables become empty. As of MySQL 4.0.18, the master replicates this effect to slaves as follows: The first time that the master uses each MEMORY table after startup, it logs an event that notifies slaves that the table must be emptied by writing a DELETE statement for that table to the binary log. See Section 13.4, “The MEMORY (HEAP) Storage Engine”, for more information about MEMORY tables.

14.7.12 Replication and Temporary Tables

Temporary tables are replicated except in the case where you shut down the slave server (not just the slave threads) and you have replicated temporary tables that are open for use in updates that have not yet been executed on the slave. If you shut down the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:

1. Issue a STOP SLAVE statement.
2. Use SHOW STATUS to check the value of the Slave_open_temp_tables [454] variable.
3. If the value is 0, issue a mysqladmin shutdown command to stop the slave.
4. If the value is not 0, restart the slave SQL thread with START SLAVE SQL_THREAD.
5. Repeat the procedure later until the Slave_open_temp_tables [454] variable is 0 and you can stop the slave.

14.7.13 Replication and User Privileges

User privileges are replicated only if the mysql database is replicated. That is, the GRANT, REVOKE, SET PASSWORD, and DROP USER (available as of MySQL 4.1.1) statements take effect on the slave only if the replication setup includes the mysql database.

If you are replicating all databases, but do not want statements that affect user privileges to be replicated, set up the slave not to replicate the mysql database, using the --replicate-wild-ignore-

`table=mysql.%` [1178] option. That option is available as of MySQL 4.0.13. The slave recognizes that privilege-related SQL statements have no effect, and thus it does not execute those statements.

14.7.14 Replication and the Query Optimizer

It is possible for the data on the master and slave to become different if a statement is written in such a way that the data modification is nondeterministic; that is, left to the will of the query optimizer. (In general, this not a good practice, even outside of replication.) For a detailed explanation of this issue, see Section B.5.8.4, “Open Issues in MySQL”.

14.7.15 Replication and Reserved Words

You can encounter problems when you attempt to replicate from an older master to a newer slave and you use identifiers on the master that are reserved words in the newer MySQL version on the slave. An example of this is using a table column named `current_user` on a 4.0 master that is replicating to a 4.1 or higher slave because `CURRENT_USER` is a reserved word beginning in MySQL 4.1. Replication can fail in such cases with Error 1064 `You have an error in your SQL syntax...`, *even if a database or table named using the reserved word or a table having a column named using the reserved word is excluded from replication*. This is due to the fact that each SQL event must be parsed by the slave prior to execution, so that the slave knows which database object or objects would be affected; only after the event is parsed can the slave apply any filtering rules defined by `--replicate-do-db` [1176], `--replicate-do-table` [1177], `--replicate-ignore-db` [1176], and `--replicate-ignore-table` [1177].

To work around the problem of database, table, or column names on the master which are regarded as reserved words by the slave, use one of the following techniques:

- Use one or more `ALTER TABLE` statements on the master to change the names of any database objects where these names would be considered reserved words on the slave, and change any SQL statements that use the old names to use the new names instead.
- In any SQL statements using these database object names, write the names as quoted identifiers using backtick characters (`).

For listings of reserved words by MySQL version, see Reserved Words, in the *MySQL Server Version Reference*. For identifier quoting rules, see Section 8.2, “Database, Table, Index, Column, and Alias Names”.

14.7.16 Slave Errors During Replication

If a statement produces the same error (identical error code) on both the master and the slave, the error is logged, but replication continues.

If a statement produces different errors on the master and the slave, the slave SQL thread terminates, and the slave writes a message to its error log and waits for the database administrator to decide what to do about the error. This includes the case that a statement produces an error on the master or the slave, but not both. To address the issue, connect to the slave manually and determine the cause of the problem. `SHOW SLAVE STATUS` is useful for this. Then fix the problem and run `START SLAVE`. For example, you might need to create a nonexistent table before you can start the slave again.

If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors` [1179] option. This option is available starting with MySQL 3.23.47.

For nontransactional storage engines such as `MyISAM`, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If

that happens on the master, the slave expects execution of the statement to result in the same error code. If it does not, the slave SQL thread stops as described previously.

14.7.17 Replication Retries and Timeouts

As of MySQL 4.1.11, there is a global system variable `slave_transaction_retries` [1181]: If the slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because it exceeded the `InnoDB` `innodb_lock_wait_timeout` [1069] or the NDBCLUSTER `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout` value, the transaction automatically retries `slave_transaction_retries` [1181] times before stopping with an error. The default value is 0 in MySQL 4.1. The total retry count can be seen in `SHOW STATUS`; see Section 5.1.5, “Server Status Variables”.

14.7.18 Replication and Time Zones

The same system time zone should be set for both master and slave. Otherwise, some statements will not be replicated properly, such as statements that use the `NOW()` [837] or `FROM_UNIXTIME()` [834] functions. You can set the time zone in which MySQL server runs by using the `--timezone=`*`timezone_name`* [245] option of the `mysqld_safe` script or by setting the `TZ` environment variable.

Starting with MySQL 4.1.3, both master and slave should use the same default connection time zone. That is, the `--default-time-zone` [386] parameter should have the same value for both master and slave. However, if the master runs MySQL 5.0 or later, this is not necessary.

`CONVERT_TZ(...,...,@@global.time_zone)` [828] is not properly replicated.

14.7.19 Replication and Transactions

Mixing transactional and nontransactional statements within the same transaction. In general, you should avoid transactions that update both transactional and nontransactional tables in a replication environment. You should also avoid using any statement that accesses both transactional and nontransactional tables and writes to any of them.

If you update transactional tables from nontransactional tables inside a `BEGIN` ... `COMMIT` sequence, updates to the binary log may be out of synchrony with table states if the nontransactional table is updated before the transaction commits. This occurs because the transaction is written to the binary log only when it is committed.

Before MySQL 4.0.15, any update to a nontransactional table is written to the binary log at once when the update is made, whereas transactional updates are written on `COMMIT` or not written at all if you use `ROLLBACK`. You must take this into account when updating both transactional tables and nontransactional tables within the same transaction. (This is true not only for replication, but also if you are using binary logging for backups.)

As of MySQL 4.0.15, we changed the logging behavior for transactions that mix updates to transactional and nontransactional tables, which solves the problems (order of statements is good in the binary log, and all needed statements are written to the binary log even in case of `ROLLBACK`). The problem that remains is that when a second connection updates the nontransactional table while the first connection transaction is not finished yet, incorrect ordering can still occur because the second connection update is written immediately after it is done.

Using different storage engines on master and slave. It is possible to replicate transactional tables on the master using nontransactional tables on the slave. For example, you can replicate an `InnoDB`

master table as a MyISAM slave table. However, there are issues that you should consider before you do this:

- There are problems if the slave is stopped in the middle of a BEGIN/COMMIT block because the slave restarts at the beginning of the BEGIN block.
- When the storage engine type of the slave is nontransactional, transactions on the master that mix updates of transactional and nontransactional tables should be avoided because they can cause inconsistency of the data between the master transactional table and the slave nontransactional table. That is, such transactions can lead to master storage engine-specific behavior with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this currently, so extra care should be taken when replicating transactional tables from the master to nontransactional ones on the slaves.

14.7.20 Replication and Variables

The foreign_key_checks [411] variable is replicated as of MySQL 4.0.14. The sql_mode [429], unique_checks [433], sql_auto_is_null [428], and storage_engine [430] (also known as table_type [431]) variables are not replicated in MySQL 4.1 or earlier.

Session variables are not replicated properly when used in statements that update tables. For example, the following sequence of statements will not insert the same data on the master and the slave:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

Update statements that refer to user-defined variables (that is, variables of the form *@var_name*) are badly replicated in 3.23 and 4.0. This is fixed in 4.1.

It is strongly recommended that you always use the same setting for the lower_case_table_names [418] system variable on both master and slave. In particular, when a case-sensitive filesystem is used, and this variable set to 1 on the slave, but to a different value on the master, names of databases are not converted to lowercase, which can cause replication to fail. This is a known issue, which is fixed in MySQL 5.6.

14.7.21 Other Replication Features

The slave can connect to the master using SSL if both are 4.1.1 or newer.

MySQL 4.1 and earlier support only replication scenarios involving one master and many slaves.

The syntax for multiple-table DELETE statements that use table aliases changed between MySQL 4.0 and 4.1. In MySQL 4.0, you should use the true table name to refer to any table from which rows should be deleted:

```
DELETE test FROM test AS t1, test2 WHERE ...
```

In MySQL 4.1, you must use the alias:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

If you use such DELETE statements, the change in syntax means that a 4.0 master cannot replicate to 4.1 (or higher) slaves.

It is safe to connect servers in a circular master/slave relationship if you use the `--log-slave-updates` [1173] option. That means that you can create a setup such as this:

```
A -> B -> C -> A
```

However, many statements do not work correctly in this kind of setup unless your client code is written to take care of the potential problems that can occur from updates that occur in different sequence on different servers.

Server IDs are encoded in binary log events, so server A knows when an event that it reads was originally created by itself and does not execute the event (unless server A was started with the `--replicate-same-server-id` [1177] option, which is meaningful only in rare cases). Thus, there are no infinite loops. This type of circular setup works only if you perform no conflicting updates between the tables. In other words, if you insert data in both A and C, you should never insert a row in A that may have a key that conflicts with a row inserted in C. You should also not update the same rows on two servers if the order in which the updates are applied is significant.

14.8 Replication and Binary Logging Options and Variables

The next few sections contain information about `mysqld` options and server variables that are used in replication and for controlling the binary log. Options and variables for use on replication masters and replication slaves are covered separately, as are options and variables relating to binary logging. A set of quick-reference tables providing basic information about these options and variables is also included (in the next section following this one).

Of particular importance is the `--server-id` [1167] option.

Command-Line Format	`--server-id=#`	
Option-File Format	`server-id`	
System Variable Name	`server_id` [426]	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 4294967295`

This option is common to both master and slave replication servers, and is used in replication to enable master and slave servers to identify themselves uniquely. This option was added in MySQL 3.23.26. For additional information, see Section 14.8.2, “Replication Master Options and Variables”, and Section 14.8.3, “Replication Slave Options and Variables”.

On the master and each slave, you must use the `server-id` [1167] option to establish a unique replication ID in the range from 1 to 2^{32} – 1. “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. Example `my.cnf` file:

```
[mysqld]
server-id=3
```

If you omit `--server-id` [1167], it assumes the default value 0, in which case a master refuses connections from all slaves, and a slave refuses to connect to a master. See Section 14.4, “How to Set Up Replication”, for more information.

14.8.1 Replication and Binary Logging Option and Variable Reference

The following tables list basic information about the MySQL command-line options and system variables applicable to replication and the binary log.

Table 14.1 Replication Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
abort-slave-event-count [1173]	Yes	Yes				
Com_change_master [450]				Yes	Both	No
Com_load_master_data [450]				Yes	Both	No
Com_load_master_table [450]				Yes	Both	No
Com_show_master_status [450]				Yes	Both	No
Com_show_new_master [450]				Yes	Both	No
Com_show_slave_hosts [450]				Yes	Both	No
Com_show_slave_status [450]				Yes	Both	No
Com_slave_start [450]				Yes	Both	No
Com_slave_stop [450]				Yes	Both	No
disconnect-slave-event-count [1173]	Yes	Yes				
init_slave [1180]	Yes	Yes	Yes		Global	Yes
log-slave-updates [1173]	Yes	Yes			Global	No
- *Variable*: log_slave_updates [1183]			Yes		Global	No
log_slave_updates [1183]	Yes	Yes	Yes		Global	No
master-connect-retry [1173]	Yes	Yes				
master-host [1174]	Yes	Yes				
master-info-file [1174]	Yes	Yes				
master-password [1174]	Yes	Yes				
master-port [1174]	Yes	Yes				
master-retry-count [1174]	Yes	Yes				
master-ssl [1174]	Yes	Yes				
master-ssl-ca [1174]	Yes	Yes				
master-ssl-capath [1174]	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
master-ssl-cert [1174]	Yes	Yes				
master-ssl-cipher [1174]	Yes	Yes				
master-ssl-key [1174]	Yes	Yes				
master-user [1174]	Yes	Yes				
relay-log [1175]	Yes	Yes			Global	No
- *Variable*: relay_log			Yes		Global	No
relay-log-index [1175]	Yes	Yes			Global	No
- *Variable*: relay_log_index			Yes		Global	No
relay_log_index	Yes	Yes	Yes		Global	No
relay-log-info-file [1175]	Yes	Yes				
- *Variable*: relay_log_info_file						
relay_log_info_file	Yes	Yes	Yes		Global	No
relay_log_purge [426]	Yes	Yes	Yes		Global	Yes
relay_log_space_limit [426]	Yes	Yes	Yes		Global	No
replicate-do-db [1176]	Yes	Yes				
replicate-do-table [1177]	Yes	Yes				
replicate-ignore-db [1176]	Yes	Yes				
replicate-ignore-table [1177]	Yes	Yes				
replicate-rewrite-db [1177]	Yes	Yes				
replicate-same-server-id [1177]	Yes	Yes				
replicate-wild-do-table [1178]	Yes	Yes				
replicate-wild-ignore-table [1178]	Yes	Yes				
report-host [1178]	Yes	Yes			Global	No
- *Variable*: report_host			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
report-password [1178]	Yes	Yes			Global	No
- *Variable*: report_password			Yes		Global	No
report-port [1179]	Yes	Yes			Global	No
- *Variable*: report_port			Yes		Global	No
report-user [1179]	Yes	Yes			Global	No
- *Variable*: report_user			Yes		Global	No
rpl_recovery_rank [1180]			Yes		Global	Yes
show-slave-auth-info [1179]	Yes	Yes				
skip-slave-start [1179]	Yes	Yes				
slave_compressed_protocol [1180]	Yes	Yes	Yes		Global	Yes
slave-load-tmpdir [1179]	Yes	Yes			Global	No
- *Variable*: slave_load_tmpdir [1180]			Yes		Global	No
slave-net-timeout [1179]	Yes	Yes			Global	Yes
- *Variable*: slave_net_timeout [1181]			Yes		Global	Yes
Slave_open_temp_tables [454]				Yes	Global	No
slave-skip-errors [1179]	Yes	Yes			Global	No
- *Variable*: slave_skip_errors [1181]			Yes		Global	No
slave_transaction_retries [1181]	Yes	Yes	Yes		Global	Yes
sql_slave_skip_counter [1181]			Yes		Global	Yes
sync_binlog [1184]	Yes	Yes	Yes		Global	Yes

Section 14.8.2, “Replication Master Options and Variables”, provides more detailed information about options and variables relating to replication master servers. For more information about options and variables relating to replication slaves, see Section 14.8.3, “Replication Slave Options and Variables”.

Table 14.2 Binary Logging Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Binlog_cache_disk_use [449]				Yes	Global	No
binlog_cache_size [1167]	Yes	Yes	Yes		Global	Yes
Binlog_cache_use [449]				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
binlog-do-db [1182]	Yes	Yes				
binlog-ignore-db [1182]	Yes	Yes				
Com_show_binlog_events [450]				Yes	Both	No
Com_show_binlogs [450]				Yes	Both	No
max_binlog_cache_size [1183]	Yes	Yes	Yes		Global	Yes
max-binlog-dump-events [1183]	Yes	Yes				
max_binlog_size [1184]	Yes	Yes	Yes		Global	Yes
sporadic-binlog-dump-fail [1183]	Yes	Yes				

Section 14.8.4, “Binary Log Options and Variables”, provides more detailed information about options and variables relating to binary logging. For additional general information about the binary log, see Section 5.3.4, “The Binary Log”.

For a table showing *all* command-line options, system and status variables used with `mysqld`, see Section 5.1.1, “Server Option and Variable Reference”.

14.8.2 Replication Master Options and Variables

This section describes the server options and system variables that you can use on replication master servers. You can specify the options either on the command line or in an option file. You can specify system variable values using `SET`.

On the master and each slave, you must use the `server-id` [1167] option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to 2^{32} – 1, and each ID must be different from every other ID in use by any other replication master or slave. Example: `server-id=3`.

For options used on the master for controlling binary logging, see Section 14.8.4, “Binary Log Options and Variables”.

14.8.3 Replication Slave Options and Variables

This section describes the server options and system variables that apply to slave replication servers. You can specify the options either on the command line or in an option file. Many of the options can be set while the server is running by using the `CHANGE MASTER TO` statement. You can specify system variable values using `SET`.

Server ID. On the master and each slave, you must use the `server-id` [1167] option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to 2^{32} – 1, and each ID must be different from every other ID. Example: `server-id=3`.

Some slave server replication options are handled in a special way, in the sense that each is ignored if a `master.info` file exists when the slave starts and contains a value for the option. The following options are handled this way:

- `--master-host` [1174]
- `--master-user` [1174]

- `--master-password` [1174]
- `--master-port` [1174]
- `--master-connect-retry` [1173]

As of MySQL 4.1.1, the following options also are handled specially:

- `--master-ssl` [1174]
- `--master-ssl-ca` [1174]
- `--master-ssl-capath` [1174]
- `--master-ssl-cert` [1174]
- `--master-ssl-cipher` [1174]
- `--master-ssl-key` [1174]

The `master.info` file format in 4.1.1 changed to include values corresponding to the SSL options. In addition, the 4.1.1 file format includes as its first line the number of lines in the file. (See Section 14.3.1, “Replication Relay and Status Files”.) If you upgrade an older server to 4.1.1, the new server upgrades the `master.info` file to the new format automatically when it starts. However, if you downgrade a 4.1.1 or newer server to a version older than 4.1.1, you should manually remove the first line before starting the older server for the first time. Note that, in this case, the downgraded server can no longer use an SSL connection to communicate with the master.

If no `master.info` file exists when the slave server starts, it uses the values for those options that are specified in option files or on the command line. This occurs when you start the server as a replication slave for the very first time, or when you have run `RESET SLAVE` and then have shut down and restarted the slave.

If the `master.info` file exists when the slave server starts, the server uses its contents and ignores any startup options that correspond to the values listed in the file. Thus, if you start the slave server with different values of the startup options that correspond to values in the `master.info` file, the different values have no effect because the server continues to use the `master.info` file. To use different values, you must either restart after removing the `master.info` file or (preferably) use the `CHANGE MASTER TO` statement to reset the values while the slave is running.

Suppose that you specify this option in your `my.cnf` file:

```
[mysqld]
master-host=some_host
```

The first time you start the server as a replication slave, it reads and uses that option from the `my.cnf` file. The server then records the value in the `master.info` file. The next time you start the server, it reads the master host value from the `master.info` file only and ignores the value in the option file. If you modify the `my.cnf` file to specify a different master host of *`some_other_host`*, the change still has no effect. You should use `CHANGE MASTER TO` instead.

Because the server gives an existing `master.info` file precedence over the startup options just described, you might prefer not to use startup options for these values at all, and instead specify them by using the `CHANGE MASTER TO` statement. See Section 12.5.2.1, “`CHANGE MASTER TO` Syntax”.

This example shows a more extensive use of startup options to configure a slave server:

```
[mysqld]
server-id=2
```

```
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
report-host=db-slave.mycompany.com
```

Startup options for replication slaves. The following list describes startup options for controlling replication slave servers. Many of these options can be set while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts. Replication-related system variables are discussed later in this section.

- `--abort-slave-event-count` [1173]

  When this option is set to some positive integer *`value`* other than 0 (the default) it affects replication behavior as follows: After the slave SQL thread has started, *`value`* log events are permitted to be executed; after that, the slave SQL thread does not receive any more events, just as if the network connection from the master were cut. The slave thread continues to run, and the output from `SHOW SLAVE STATUS` displays `Yes` in both the `Slave_IO_Running` and the `Slave_SQL_Running` columns, but no further events are read from the relay log.

  This option is used internally by the MySQL test suite for replication testing and debugging. It is not intended for use in a production setting.

- `--disconnect-slave-event-count` [1173]

  This option is used internally by the MySQL test suite for replication testing and debugging.

- `--log-slave-updates` [1173]

  Normally, a slave does not log to its own binary log any updates that are received from a master server. This option tells the slave to log the updates performed by its SQL thread to its own binary log. For this option to have any effect, the slave must also be started with the `--log-bin` [1181] option to enable binary logging. `--log-slave-updates` [1173] is used when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

  ```
  A -> B -> C
  ```

  Here, `A` serves as the master for the slave `B`, and `B` serves as the master for the slave `C`. For this to work, `B` must be both a master *and* a slave. You must start both `A` and `B` with `--log-bin` [1181] to enable binary logging, and `B` with the `--log-slave-updates` [1173] option so that updates received from `A` are logged by `B` to its binary log.

- `--log-warnings[=`*`level`*`]` [389]

  This option causes a server to print more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network/connection failure, and informs you as to how each slave thread started. This option is enabled (1) by default as of MySQL 4.0.19 and 4.1.2; to disable it, use `--log-warnings=0` [389]. As of MySQL 4.0.21 and 4.1.3, aborted connections are not logged to the error log unless the value is greater than 1.

  Note that the effects of this option are not limited to replication. It produces warnings across a spectrum of server activities.

- `--master-connect-retry=`*`seconds`* [1173]

  The number of seconds that the slave thread sleeps before trying to reconnect to the master in case the master goes down or the connection is lost. The value in the `master.info` file takes precedence

if it can be read. If not set, the default is 60. Connection retries are not invoked until the slave times out reading data from the master according to the value of `--slave-net-timeout` [1179]. The number of reconnection attempts is limited by the `--master-retry-count` [1174] option.

- `--master-host=host_name` [1174]

  The host name or IP address of the master replication server. The value in `master.info` takes precedence if it can be read. If no master host is specified, the slave thread does not start.

- `--master-info-file=file_name` [1174]

  The name to use for the file in which the slave records information about the master. The default name is `master.info` in the data directory. For information about the format of this file, see Section 14.3.3, "The Slave Status Files".

- `--master-password=password` [1174]

  The password of the account that the slave thread uses for authentication when it connects to the master. The value in the `master.info` file takes precedence if it can be read. If not set, an empty password is assumed.

- `--master-port=port_number` [1174]

  The TCP/IP port number that the master is listening on. The value in the `master.info` file takes precedence if it can be read. If not set, the compiled-in setting is assumed (normally 3306).

- `--master-retry-count=count` [1174]

  The number of times that the slave tries to connect to the master before giving up. Reconnects are attempted at intervals set by the `CHANGE MASTER TO` statement or `--master-connect-retry` [1173] option and reconnects are triggered when data reads by the slave time out according to the `--slave-net-timeout` [1179] option. The default value is 86400.

- `--master-ssl` [1174], `--master-ssl-ca=file_name`, `--master-ssl-capath=directory_name`, `--master-ssl-cert=file_name`, `--master-ssl-cipher=cipher_list`, `--master-ssl-key=file_name`

  These options are used for setting up a secure replication connection to the master server using SSL. Their meanings are the same as the corresponding `--ssl` [521], `--ssl-ca` [522], `--ssl-capath` [522], `--ssl-cert` [522], `--ssl-cipher` [522], `--ssl-key` [522] options that are described in Section 5.6.6.3, "SSL Command Options". The values in the `master.info` file take precedence if they can be read.

  These options are operational as of MySQL 4.1.1.

- `--master-user=user_name` [1174]

  The user name of the account that the slave thread uses for authentication when it connects to the master. This account must have the `REPLICATION SLAVE` [492] privilege. `FILE` [491] privilege instead.) The value in the `master.info` file takes precedence if it can be read. If the master user name is not set, the name `test` is assumed.

- `--read-only` [1174]

  Cause the slave to permit no updates except from slave threads or from users having the `SUPER` [493] privilege. On a slave server, this can be useful to ensure that the slave accepts updates only from its master server and not from clients. This variable does not apply to `TEMPORARY` tables.

This option is available as of MySQL 4.0.14.

- `--relay-log=file_name` [1175]

  The basename for the relay log. The default basename is `host_name-relay-bin`. The server writes the file in the data directory unless the basename is given with a leading absolute path name to specify a different directory. The server creates relay log files in sequence by adding a numeric suffix to the basename.

  Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the `--relay-log` [1175] option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.3, “Specifying Program Options”.

  If you specify this option, the value specified is also used as the basename for the relay log index file. You can override this behavior by specifying a different relay log index file basename using the `--relay-log-index` [1175] option.

  You may find the `--relay-log` [1175] option useful in performing the following tasks:

  - Creating relay logs whose names are independent of host names.
  - If you need to put the relay logs in some area other than the data directory because your relay logs tend to be very large and you do not want to decrease `max_relay_log_size` [420].
  - To increase speed by using load-balancing between disks.

- `--relay-log-index=file_name` [1175]

  The name to use for the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where `host_name` is the name of the slave server.

  Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the `--relay-log-index` [1175] option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.3, “Specifying Program Options”.

  If you specify this option, the value specified is also used as the basename for the relay logs. You can override this behavior by specifying a different relay log file basename using the `--relay-log` [1175] option.

- `--relay-log-info-file=file_name` [1175]

  The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory. For information about the format of this file, see Section 14.3.3, “The Slave Status Files”.

- `--relay-log-purge={0|1}` [1175]

  Disable or enable automatic purging of relay logs as soon as they are no longer needed. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = N`.

This option is available as of MySQL 4.1.1.

- `--max-relay-log-size=size` [1176]

  The size at which the server rotates relay log files automatically. For more information, see Section 14.3.1, “Replication Relay and Status Files”. Default is 1GB.

  This option is available as of MySQL 4.0.14.

- `--relay-log-space-limit=size` [1176]

  This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means “no limit.” This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs because not doing so would cause a deadlock (which is what happens before MySQL 4.0.13). You should not set `--relay-log-space-limit` [1176] to less than twice the value of `--max-relay-log-size` [1176] (or `--max-binlog-size` [1184] if `--max-relay-log-size` [1176] is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` [1176] is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to ignore `--relay-log-space-limit` [1176] temporarily.

- `--replicate-do-db=db_name` [1176]

  Tell the slave SQL thread to restrict replication to statements where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database. Note that this does not replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while having selected a different database or no database.

  Warning

  To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

  An example of what does not work as you might expect: If the slave is started with `--replicate-do-db=sales` [1176] and you issue the following statements on the master, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

  The main reason for this “check just the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` or multiple-table `UPDATE` statements that go across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

  If you need cross-database updates to work, make sure that you have MySQL 3.23.28 or later, and use `--replicate-wild-do-table=db_name.%` [1178] instead. See Section 14.9, “How Servers Evaluate Replication Filtering Rules”.

- `--replicate-ignore-db=db_name` [1176]

Tells the slave SQL thread not to replicate any statement where the default database (that is, the one selected by USE) is *db_name*. To specify more than one database to ignore, use this option multiple times, once for each database. You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See Section 14.9, “How Servers Evaluate Replication Filtering Rules”.

An example of what does not work as you might expect: If the slave is started with --replicate-ignore-db=sales [1176] and you issue the following statements on the master, the UPDATE statement *is* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

Note

In the preceding example the statement is replicated because --replicate-ignore-db [1176] only applies to the default database (set through the USE statement). Because the sales database was specified explicitly in the statement, the statement has not been filtered.

If you need cross-database updates to work, use --replicate-wild-ignore-table=*db_name*.% [1178] instead. See Section 14.9, “How Servers Evaluate Replication Filtering Rules”.

- --replicate-do-table=*db_name.tbl_name* [1177]

  Tells the slave SQL thread to restrict replication to the specified table. To specify more than one table, use this option multiple times, once for each table. This works for both cross-database updates and default database updates, in contrast to --replicate-do-db [1176]. See Section 14.9, “How Servers Evaluate Replication Filtering Rules”.

- --replicate-ignore-table=*db_name.tbl_name* [1177]

  Tells the slave SQL thread not to replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to --replicate-ignore-db [1176]. See Section 14.9, “How Servers Evaluate Replication Filtering Rules”.

- --replicate-rewrite-db=*from_name->to_name* [1177]

  Tells the slave to translate the default database (that is, the one selected by USE) to *to_name* if it was *from_name* on the master. Only statements involving tables are affected (not statements such as CREATE DATABASE, DROP DATABASE, and ALTER DATABASE), and only if *from_name* is the default database on the master. This does not work for cross-database updates. To specify multiple rewrites, use this option multiple times. The server uses the first one with a *from_name* value that matches. The database name translation is done *before* the --replicate-* rules are tested.

  If you use this option on the command line and the “>” character is special to your command interpreter, quote the option value. For example:

  ```
  shell> mysqld --replicate-rewrite-db="olddb->newdb"
  ```

- --replicate-same-server-id [1177]

  To be used on slave servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the slave does not skip events having its own server ID.

Normally, this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` [1173] is used. Be careful that starting from MySQL 4.1, by default the slave I/O thread does not even write binary log events to the relay log if they have the slave's server id (this optimization helps save disk usage compared to 4.0). So if you want to use `--replicate-same-server-id` [1177] in 4.1 versions, be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=`*`db_name.tbl_name`* [1178]

  Tells the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the "`%`" and "`_`" wildcard characters, which have the same meaning as for the `LIKE` [804] pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See Section 14.9, "How Servers Evaluate Replication Filtering Rules".

  Example: `--replicate-wild-do-table=foo%.bar%` [1178] replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

  If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%` [1178], database-level statements are replicated if the database name matches the pattern `foo%`.

  To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate tables from the `my1ownAABCdb` database, you should escape the "`_`" and "`%`" characters like this: `--replicate-wild-do-table=my\_own\%db` [1178]. If you use the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my\\_own\\%db` [1178].

- `--replicate-wild-ignore-table=`*`db_name.tbl_name`* [1178]

  Tells the slave thread not to replicate a statement where any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See Section 14.9, "How Servers Evaluate Replication Filtering Rules".

  Example: `--replicate-wild-ignore-table=foo%.bar%` [1178] does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

  For information about how matching works, see the description of the `--replicate-wild-do-table` [1178] option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` [1178] as well.

- `--report-host=`*`host_name`* [1178]

  The host name or IP address of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP address of the slave from the TCP/IP socket after the slave connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

  This option is available as of MySQL 4.0.0.

- `--report-password=`*`password`* [1178]

The account password of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` [1179] option is given.

- `--report-port=slave_port_num` [1179]

  The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a nondefault port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

  This option is available as of MySQL 4.0.0.

- `--report-user=user_name` [1179]

  The account user name of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` [1179] option is given.

- `--show-slave-auth-info` [1179]

  Display slave user names and passwords in the output of `SHOW SLAVE HOSTS` on the master server for slaves started with the `--report-user` [1179] and `--report-password` [1178] options.

- `--skip-slave-start` [1179]

  Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}` [1180]

  If this option is set to 1, use compression for the slave/master protocol if both the slave and the master support it. The default is 0 (no compression).

- `--slave-load-tmpdir=file_name` [1179]

  The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` [432] system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `--relay-log` [1175] option to place the relay logs in that file system.

  The directory specified by this option should be located in a disk-based file system (not a memory-based file system) because the temporary files used to replicate `LOAD DATA INFILE` must survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process.

- `--slave-net-timeout=seconds` [1179]

  The number of seconds to wait for more data from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `CHANGE MASTER TO` statement or `--master-connect-retry` [1173] option and the number of reconnection attempts is limited by the `--master-retry-count` [1174] option. The default is 3600 seconds (one hour).

- `--slave-skip-errors=[err_code1,err_code2,...|all]` [1179]

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. Appendix B, *Errors, Error Codes, and Common Problems*, lists server error codes.

You can also (but should not) use the very nonrecommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned*.

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

System variables used on replication slaves. The following list describes system variables for controlling replication slave servers. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used with replication slaves are listed earlier in this section.

- `init_slave` [1180]

  This variable is similar to `init_connect` [414], but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` [414] variable.

  Note

  The SQL thread sends an acknowledgment to the client before it executes `init_slave` [1180]. Therefore, it is not guaranteed that `init_slave` [1180] has been executed when `START SLAVE` returns. See Section 12.5.2.7, “START SLAVE Syntax”, for more information.

  This variable was added in MySQL 4.1.2.

- `rpl_recovery_rank` [1180]

  This variable is unused.

- `slave_compressed_protocol` [1180]

  Whether to use compression of the master/slave protocol if both the slave and the master support it. This variable was added in MySQL 4.0.3.

- `slave_load_tmpdir` [1180]

  The name of the directory where the slave creates temporary files for replicating `LOAD DATA INFILE` statements. This variable was added in MySQL 4.0.0.

- `slave_net_timeout` [1181]

  The number of seconds to wait for more data from a master/slave connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made using Unix socket files, named pipes, or shared memory. This variable was added in MySQL 3.23.40.

- `slave_skip_errors` [1181]

  Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This variable tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value. This variable was added in MySQL 3.23.47.

- `slave_transaction_retries` [1181]

  If a replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because the transaction's execution time exceeded `InnoDB`'s `innodb_lock_wait_timeout` [1069] or `NDBCLUSTER`'s `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`, it automatically retries `slave_transaction_retries` [1181] times before stopping with an error. The default in MySQL 4.1 is `0`. You must explicitly set the value to greater than 0 to enable the “retry” behavior, which is often desirable.

- `sql_slave_skip_counter` [1181]

  The number of events from the master that a slave server should skip. This variable was added in MySQL 3.23.33.

Important

If skipping the number of events specified by setting this variable would cause the slave to begin in the middle of an event group, the slave continues to skip until it finds the beginning of the next event group and begins from that point. For more information, see Section 12.5.2.6, “`SET GLOBAL SQL_SLAVE_SKIP_COUNTER` Syntax”.

14.8.4 Binary Log Options and Variables

You can use the `mysqld` options and system variables that are described in this section to affect the operation of the binary log as well as to control which statements are written to the binary log. For additional information about the binary log, see Section 5.3.4, “The Binary Log”. For additional information about using MySQL server options and system variables, see Section 5.1.2, “Server Command Options”, and Section 5.1.3, “Server System Variables”.

Startup options used with binary logging. The following list describes startup options for enabling and configuring the binary log. System variables used with binary logging are discussed later in this section.

- `--log-bin[=`*`base_name`*`]` [1181]

  Enable binary logging. The server logs all statements that change data to the binary log, which is used for backup and replication. See Section 5.3.4, “The Binary Log”.

  The option value, if given, is the basename for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the basename. It is recommended that you specify a basename (see Section B.5.8.4, “Open Issues in MySQL”, for the reason). Otherwise, MySQL uses *`host_name`*`-bin` as the basename.

- `--log-bin-index[=file_name]` [1182]

  The index file for binary log file names. See Section 5.3.4, “The Binary Log”. If you omit the file name, and if you did not specify one with `--log-bin` [1181], MySQL uses `host_name-bin.index` as the file name.

Statement selection options. The options in the following list affect which statements are written to the binary log, and thus sent by a replication master server to its slaves. There are also options for slave servers that control which statements received from the master should be executed or ignored. For details, see Section 14.8.3, “Replication Slave Options and Variables”.

- `--binlog-do-db=db_name` [1182]

  This option affects binary logging in a manner similar to the way that `--replicate-do-db` [1176] affects replication.

  Tell the server to restrict binary logging to updates for which the default database is *db_name* (that is, the database selected by `USE`). All other databases that are not explicitly mentioned are ignored. If you use this option, you should ensure that you do updates only in the default database.

  There is an exception to this for `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

  An example of what does not work as you might expect: If the server is started with `--binlog-do-db=sales` [1182] and you issue the following statements, the `UPDATE` statement is *not* logged:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

  The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

  Another case which may not be self-evident occurs when a given database is replicated even though it was not specified when setting the option. If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though `prices` was not included when setting `--binlog-do-db`:

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

  Because `sales` is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

  Important

  To log multiple databases, use this option multiple times, specifying the option once for each database to be logged. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

- `--binlog-ignore-db=db_name` [1182]

  This option affects binary logging in a manner similar to the way that `--replicate-ignore-db` [1176] affects replication.

Tell the server to suppress binary logging of updates for which the default database is *db_name* (that is, the database selected by USE). If you use this option, you should ensure that you do updates only in the default database.

As with the --binlog-do-db [1182] option, there is an exception for the CREATE DATABASE, ALTER DATABASE, and DROP DATABASE statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with binlog-ignore-db=sales, and you run USE prices; UPDATE sales.january SET amount = amount + 1000;, this statement *is* written into the binary log.

Important

To ignore multiple databases, use this option multiple times, specifying the option once for each database to be ignored. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

Testing and debugging options. The following binary log options are used in replication testing and debugging. They are not intended for use in normal operations.

- --max-binlog-dump-events=*N* [1183]

  This option is used internally by the MySQL test suite for replication testing and debugging.

- --sporadic-binlog-dump-fail [1183]

  This option is used internally by the MySQL test suite for replication testing and debugging.

System variables used with the binary log. The following list describes system variables for controlling binary logging. They can be set at server startup and some of them can be changed at runtime using SET. Server options used to control binary logging are listed earlier in this section.

- log_bin [1183]

System Variable Name	log_bin [1183]
Variable Scope	Global
Dynamic Variable	No

Whether the binary log is enabled. If the --log-bin [1181] option is used, then the value of this variable is ON; otherwise it is OFF. This variable reports only on the status of binary logging (enabled or disabled); it does not actually report the value to which --log-bin [1181] is set.

See Section 5.3.4, “The Binary Log”.

- log_slave_updates [1183]

  Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this variable to have any effect. This variable was added in MySQL 3.23.17. See Section 14.8.3, “Replication Slave Options and Variables”.

- max_binlog_cache_size [1183]

  If a multiple-statement transaction requires more than this many bytes of memory, the server generates a Multi-statement transaction required more than 'max_binlog_cache_size'

bytes of storage error. The minimum value is 4096; the maximum and default values are 4GB on 32-bit platforms and 16 PB (petabytes) on 64-bit platforms. This variable was added in MySQL 3.23.29.

- max_binlog_size [1184]

  If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). The minimum value is 4096 bytes. The maximum and default value is 1GB.

  A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary log files larger than max_binlog_size [1184].

  If max_relay_log_size [420] is 0, the value of max_binlog_size [1184] applies to relay logs as well.

- sync_binlog [1184]

  If the value of this variable is greater than 0, the MySQL server synchronizes its binary log to disk (using fdatasync()) after every sync_binlog [1184] writes to the binary log. There is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value of sync_binlog [1184] is 0, which does no synchronizing to disk. A value of 1 is the safest choice because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast). This variable was added in MySQL 4.1.3.

  If the value of sync_binlog [1184] is 0 (the default), no extra flushing is done. The server relies on the operating system to flush the file contents occasionally as for any other file.

14.9 How Servers Evaluate Replication Filtering Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master, you can control which databases to log changes for by using the --binlog-do-db [1182] and --binlog-ignore-db [1182] options to control binary logging. For a description of the rules that servers use in evaluating these options, see Section 14.9.1, “Evaluation of Database-Level Replication and Binary Logging Options”. You should not use these options to control which databases and tables are replicated. Instead, use filtering on the slave to control the events that are executed on the slave.

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the --replicate-* options that the slave was started with. (See Section 14.8, “Replication and Binary Logging Options and Variables”.)

In the simplest case, when there are no --replicate-* options, the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given.

Database-level options (--replicate-do-db [1176], --replicate-ignore-db [1176]) are checked first; see Section 14.9.1, “Evaluation of Database-Level Replication and Binary Logging Options”, for a description of this process. If no matching database-level options are found, option checking proceeds to any table-level options that may be in use, as discussed in Section 14.9.2, “Evaluation of Table-Level Replication Options”.

To make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and nonwildcard options. An example of the latter that may

have unintended effects is the use of `--replicate-do-db` [1176] and `--replicate-wild-do-table` [1178] together, where `--replicate-wild-do-table` [1178] uses a pattern for the database name that matches the name given for `--replicate-do-db` [1176]. Suppose a replication slave is started with `--replicate-do-db=dbx` [1176] `--replicate-wild-do-table=db%.t1` [1178]. Then, suppose that on the master, you issue the statement `CREATE DATABASE dbx`. Although you might expect it, this statement is not replicated because it does not reference a table named `t1`.

If any `--replicate-rewrite-db` [1177] options were specified, they are applied before the `--replicate-*` filtering rules are tested.

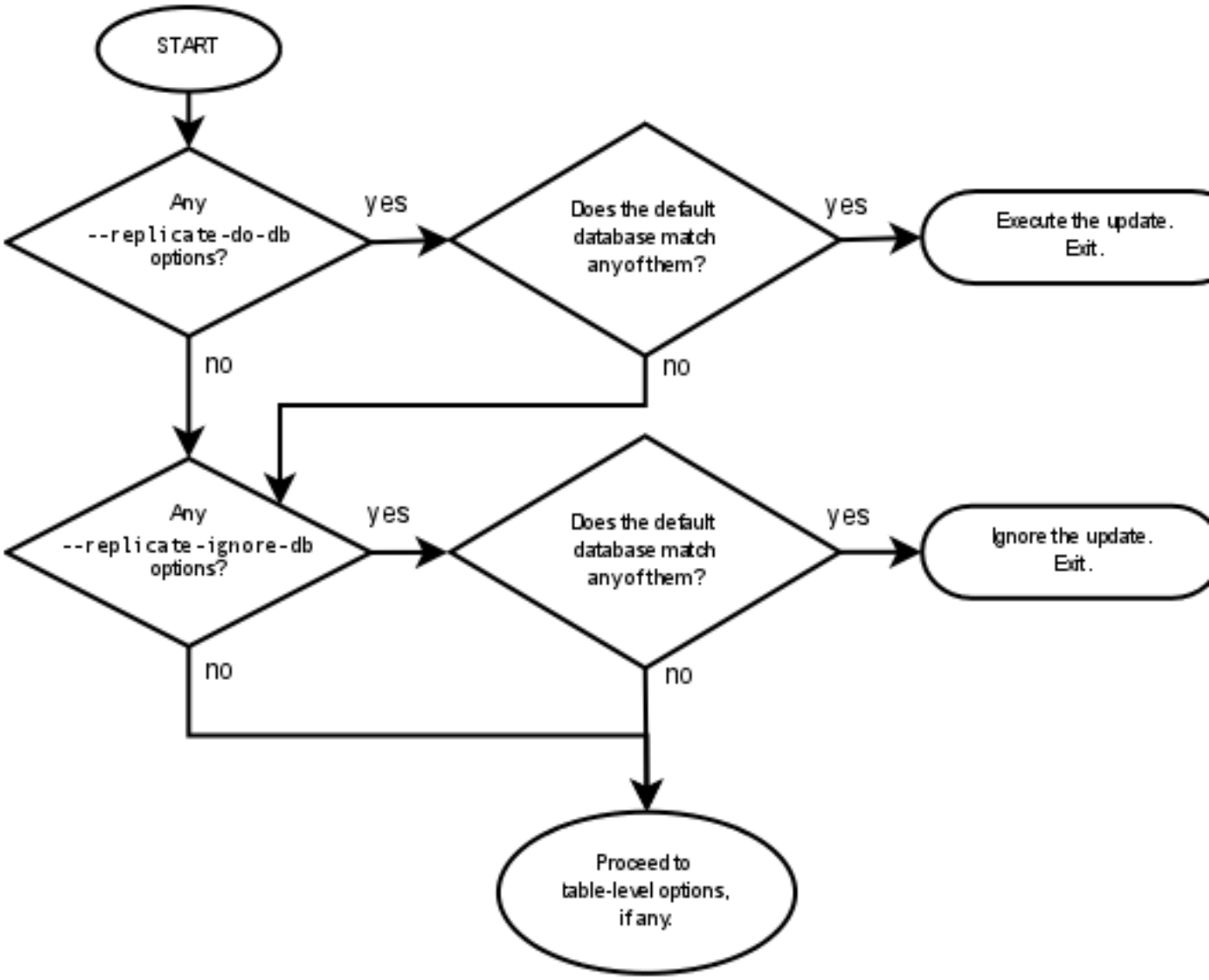
Note

In MySQL 4.1, database-level filtering options are case-sensitive on platforms supporting case sensitivity in filenames, whereas table-level filtering options are not. This is true regardless of the value of the `lower_case_table_names` [418] system variable.

14.9.1 Evaluation of Database-Level Replication and Binary Logging Options

When evaluating replication options, the slave begins by checking to see whether there are any `--replicate-do-db` [1176] or `--replicate-ignore-db` [1176] options that apply. When using `--binlog-do-db` [1182] or `--binlog-ignore-db` [1182], the process is similar, but the options are checked on the master.

The checking of the database-level options proceeds as shown in the following diagram.



The steps involved are listed here:

1. Are there any `--replicate-do-db` [1176] options?
   - **Yes.** Do any of them match the database?
     - **Yes.** Execute the statement and exit.
     - **No.** Continue to step 2.
   - **No.** Continue to step 2.
2. Are there any `--replicate-ignore-db` [1176] options?
   - **Yes.** Do any of them match the database?
     - **Yes.** Ignore the statement and exit.

- **No.** Continue to step 3.

- **No.** Continue to step 3.

3. Proceed to checking the table-level replication options, if there are any. For a description of how these options are checked, see Section 14.9.2, “Evaluation of Table-Level Replication Options”.

Important

A statement that is still permitted at this stage is not yet actually executed. The statement is not executed until all table-level options (if any) have also been checked, and the outcome of that process permits execution of the statement.

For binary logging, the steps involved are listed here:

1. Are there any `--binlog-do-db` [1182] or `--binlog-ignore-db` [1182] options?

   - **Yes.** Continue to step 2.

   - **No.** Log the statement and exit.

2. Is there a default database (has any database been selected by `USE`)?

   - **Yes.** Continue to step 3.

   - **No.** Ignore the statement and exit.

3. There is a default database. Are there any `--binlog-do-db` [1182] options?

   - **Yes.** Do any of them match the database?

     - **Yes.** Log the statement and exit.

     - **No.** Ignore the statement and exit.

   - **No.** Continue to step 4.

4. Do any of the `--binlog-ignore-db` [1182] options match the database?

   - **Yes.** Ignore the statement and exit.

   - **No.** Log the statement and exit.

Important

An exception is made in the rules just given for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. In those cases, the database being *created, altered, or dropped* replaces the default database when determining whether to log or to ignore updates.

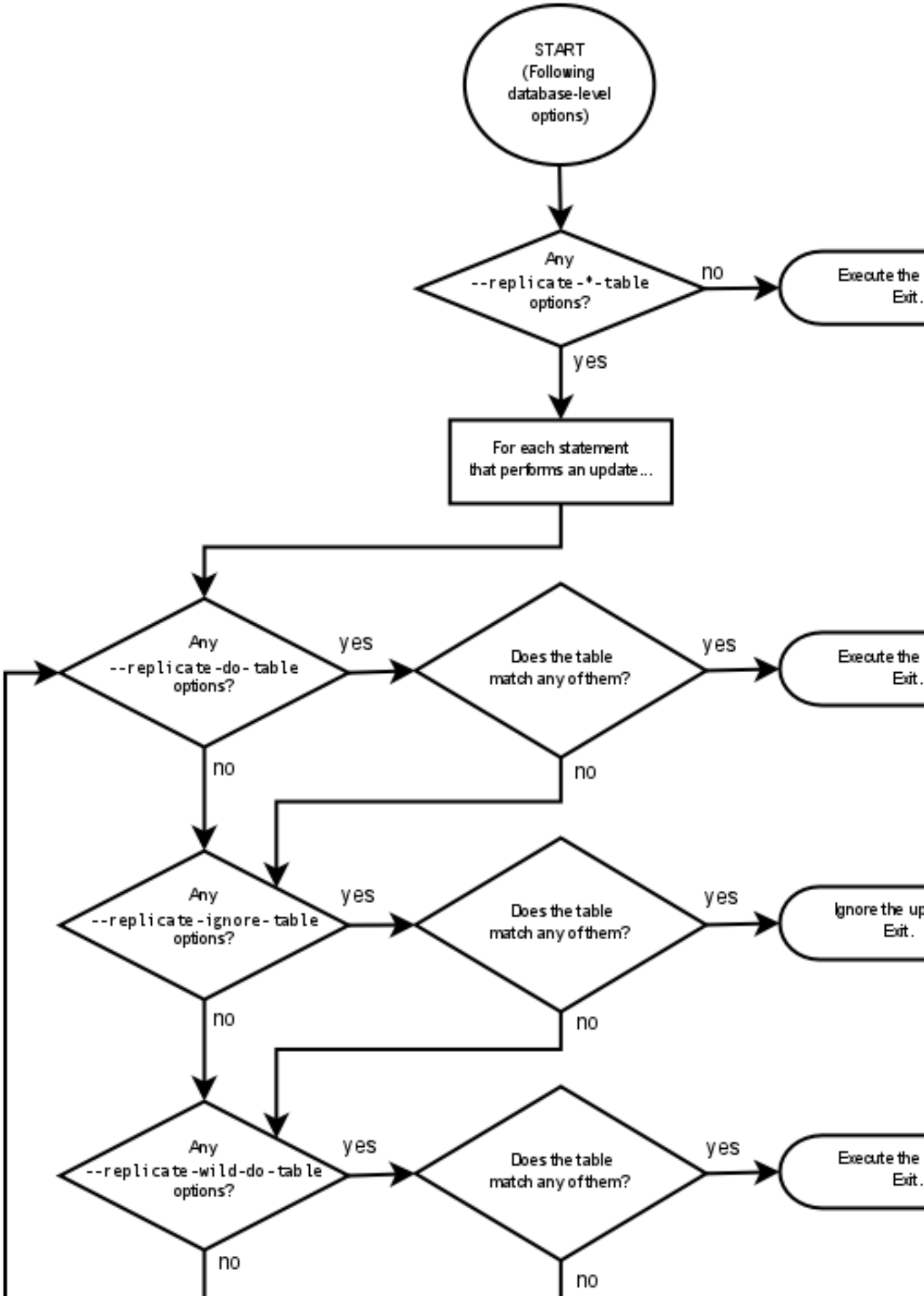
`--binlog-do-db` [1182] can sometimes mean “ignore other databases”. For example, a server running with only `--binlog-do-db=sales` [1182] does not write to the binary log statements for which the default database differs from `sales`.

14.9.2 Evaluation of Table-Level Replication Options

The slave checks for and evaluates table options only if no matching database options were found (see Section 14.9.1, “Evaluation of Database-Level Replication and Binary Logging Options”).

First, as a preliminary condition, the slave checks whether the statement occurs within a stored function, in which case the slave executes the statement and exits.

Having reached this point, if there are no table options, the slave simply executes all statements. If there are any `--replicate-do-table` [1177] or `--replicate-wild-do-table` [1178] options, the statement must match one of these if it is to be executed; otherwise, it is ignored. If there are any `--replicate-ignore-table` [1177] or `--replicate-wild-ignore-table` [1178] options, all statements are executed except those that match any of these options. This process is illustrated in the following diagram.


START
(Following
database-level
options)
Any
--replicate-*-table
options?
no
Execute the
Exit.
yes
For each statement
that performs an update...
Any
--replicate-do-table
options?
yes
Does the table
match any of them?
yes
Execute the
Exit.
no
no
Any
--replicate-ignore-table
options?
yes
Does the table
match any of them?
yes
Ignore the up
Exit.
no
no
Any
--replicate-wild-do-table
options?
yes
Does the table
match any of them?
yes
Execute the
Exit.
no
no

The following steps describe this evaluation in more detail:

1. Are there any table options?
   - **Yes.** Continue to step 2.
   - **No.** Execute the statement and exit.
2. Are there any `--replicate-do-table` [1177] options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Execute the statement and exit.
     - **No.** Continue to step 3.
   - **No.** Continue to step 3.
3. Are there any `--replicate-ignore-table` [1177] options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Ignore the statement and exit.
     - **No.** Continue to step 4.
   - **No.** Continue to step 4.
4. Are there any `--replicate-wild-do-table` [1178] options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Execute the statement and exit.
     - **No.** Continue to step 5.
   - **No.** Continue to step 5.
5. Are there any `--replicate-wild-ignore-table` [1178] options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Ignore the statement and exit.
     - **No.** Continue to step 6.
   - **No.** Continue to step 6.
6. Are there any `--replicate-do-table` [1177] or `--replicate-wild-do-table` [1178] options?
   - **Yes.** Ignore the statement and exit.
   - **No.** Execute the statement and exit.

14.9.3 Replication Rule Application

This section provides additional explanation and examples of usage for different combinations of replication filtering options.

Some typical combinations of replication filter rule types are given in the following table:

Condition (Types of Options)	Outcome
No `--replicate-*` options at all:	The slave executes all events that it receives from the master.
`--replicate-*-db` options, but no table options:	The slave accepts or ignores statements using the database options. It executes all statements permitted by those options because there are no table restrictions.
`--replicate-*-table` options, but no database options:	All statements are accepted at the database-checking stage because there are no database conditions. The slave executes or ignores statements based solely on the table options.
A combination of database and table options:	The slave accepts or ignores statements using the database options. Then it evaluates all statements permitted by those options according to the table options. This can sometimes lead to results that seem counterintuitive; see the text for an example.

A more complex example follows.

Suppose that we have two tables `mytbl1` in database `db1` and `mytbl2` in database `db2` on the master, and the slave is running with the following options (and no other replication filtering options):

```
replicate-ignore-db = db1
replicate-do-table  = db2.tbl2
```

Now we execute the following statements on the master:

```
USE db1;
INSERT INTO db2.tbl2 VALUES (1);
```

The outcome may not match initial expectations, because the `USE` statement causes `db1` to be the default database. Thus the `--replicate-ignore-db` [1176] option matches, which causes the `INSERT` statement to be ignored. Because there was a match with a database-level option, the table options are not checked; processing immediately moves to the next statement executed on the master.

14.10 Replication FAQ

Q: How do I configure a slave if the master is running and I do not want to stop it?

A: There are several possibilities. If you have taken a snapshot backup of the master at some point and recorded the binary log file name and offset (from the output of `SHOW MASTER STATUS`) corresponding to the snapshot, use the following procedure:

1. Make sure that the slave is assigned a unique server ID.
2. Execute the following statement on the slave, filling in appropriate values for each option:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='master_user_name',
    ->     MASTER_PASSWORD='master_pass',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

3. Execute `START SLAVE` on the slave.

If you do not have a backup of the master server, here is a quick procedure for creating one. All steps should be performed on the master host.

1. Issue this statement to acquire a global read lock:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

2. With the lock still in place, execute this command (or a variation of it):

```
shell> tar zcf /tmp/backup.tar.gz /var/lib/mysql
```

3. Issue this statement and record the output, which you will need later:

```
mysql> SHOW MASTER STATUS;
```

4. Release the lock:

```
mysql> UNLOCK TABLES;
```

An alternative to using the preceding procedure to make a binary copy is to make an SQL dump of the master. To do this, you can use `mysqldump --master-data` on your master and later load the SQL dump into your slave. However, this is slower than making a binary copy.

Regardless of which of the two methods you use, afterward follow the instructions for the case when you have a snapshot and have recorded the log file name and offset. You can use the same snapshot to set up several slaves. Once you have the snapshot of the master, you can wait to set up a slave as long as the binary logs of the master are left intact. The two practical limitations on the length of time you can wait are the amount of disk space available to retain binary logs on the master and the length of time it takes the slave to catch up.

Q: Does the slave need to be connected to the master all the time?

A: No, it does not. The slave can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in synchrony with the master unless you take some special measures.

Q: How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?

A: If the slave is 4.1.1 or newer, read the `Seconds_Behind_Master` column in `SHOW SLAVE STATUS`, which shows the number of seconds that the slave SQL thread is behind processing the master binary log. A high number (or an increasing one) can indicate that the slave is unable to cope with the large number of queries from the master.

A value of 0 for `Seconds_Behind_Master` can usually be interpreted as meaning that the slave has caught up with the master, but there are some cases where this is not strictly true. For example, this can occur if the network connection between master and slave is broken but the slave I/O thread has not yet noticed this—that is, `slave_net_timeout` [1181] has not yet elapsed.

It is also possible that transient values for `Seconds_Behind_Master` may not reflect the situation accurately. When the slave SQL thread has caught up on I/O, `Seconds_Behind_Master` displays 0; but when the slave I/O thread is still queuing up a new event, `Seconds_Behind_Master` may show a large value until the SQL thread finishes executing the new event. This is especially likely when the events have

old timestamps; in such cases, if you execute `SHOW SLAVE STATUS` several times in a relatively short peiod, you may see this value change back and forth repeatedly between 0 and a relatively large value.

For versions of MySQL prior to 4.1.1, it is possible to determine how far behind the slave is only if `SHOW SLAVE STATUS` on the slave shows that the SQL thread is running (or for MySQL 3.23, that the slave thread is running), and that the thread has executed at least one event from the master. See Section 14.3, “Replication Implementation Details”.

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see `Time` values like 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This is because the slave is executing statements that are one hour old.

Q: How do I force the master to block updates until the slave catches up?

A: Use the following procedure:

1. On the master, execute these statements:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SHOW MASTER STATUS;
   ```

   Record the replication coordinates (the log file name and offset) from the output of the `SHOW` statement.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` [879] function are the replication coordinate values obtained in the previous step:

   ```
   mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
   ```

   The `SELECT` statement blocks until the slave reaches the specified log file and offset. At that point, the slave is in synchrony with the master and the statement returns.

3. On the master, issue the following statement to enable the master to begin processing updates again:

   ```
   mysql> UNLOCK TABLES;
   ```

Q: What issues should I be aware of when setting up two-way replication?

A: MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make an update to co-master 2 that makes the update of client A work differently than it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different from what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention, because the

updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

Q: How can I use replication to improve performance of my system?

A: You should set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rackspace for, and distribute the reads among the master and the slaves. You can also start the slaves with the `--skip-innodb` [1066], `--skip-bdb` [392], `--low-priority-updates` [389], and `--delay-key-write=ALL` [386] options to get speed improvements on the slave end. In this case, the slave uses nontransactional `MyISAM` tables instead of `InnoDB` and `BDB` tables to get more speed by eliminating transactional overhead.

Q: What should I do to prepare client code in my own applications to use performance-enhancing replication?

A: If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to it clean up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions; for example, to log how long each statement took, or which statement among those issued gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the `replace` utility that comes with standard MySQL distributions, or just write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

Q: When and how much can MySQL replication improve the performance of my system?

A: MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

To determine how many slaves you can use before the added benefits begin to level out, and how much you can improve performance of your site, you need to know your query patterns, and to determine empirically by benchmarking the relationship between the throughput for reads (reads per second, or `reads`) and for writes (`writes`) on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is 1200 – 2 × `writes`. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Suppose that the master and each slave have the same capacity, and that we have one master and *N* slaves. Then we have for each server (master or slave):

`reads` = 1200 – 2 × `writes`

`reads` = 9 × `writes` / (*N* + 1) (reads are split, but writes replicated to all slaves)

9 × `writes` / (*N* + 1) + 2 × `writes` = 1200

`writes` = 1200 / (2 + 9/(*N* + 1))

The last equation indicates the maximum number of writes for *N* slaves, given a maximum possible read rate of 1,200 per minute and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If *N* = 0 (which means we have no replication), our system can handle about 1200/11 = 109 writes per second.
- If *N* = 1, we get up to 184 writes per second.
- If *N* = 8, we get up to 400 writes per second.
- If *N* = 17, we get up to 480 writes per second.
- Eventually, as *N* approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

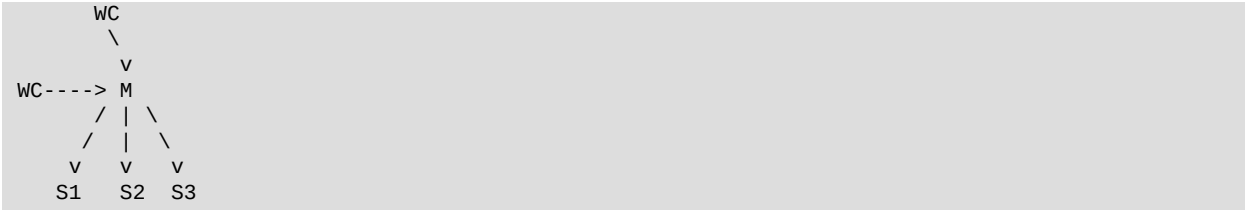
Note that these computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add *N* replication slaves. However, answering the following questions should help you decide whether and by how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

Q: How can I use replication to provide redundancy or high availability?

A: With the currently available features, you would have to set up a master and a slave (or several slaves), and to write a script that monitors the master to check whether it is up. Then instruct your applications and the slaves to change master in case of failure. Some suggestions:

- To tell a slave to change its master, use the `CHANGE MASTER TO` statement.
- A good way to keep your applications informed as to the location of the master is by having a dynamic DNS entry for the master. With `bind` you can use `nsupdate` to dynamically update your DNS.
- Run your slaves with the `--log-bin` [1181] option and without `--log-slave-updates` [1173]. In this way, the slave is ready to become a master as soon as you issue `STOP SLAVE`; `RESET MASTER`, and `CHANGE MASTER TO` statement on the other slaves. For example, assume that you have the following setup:



In this diagram, `M` means the master, `S` the slaves, `WC` the clients issuing database writes and reads; clients that issue only database reads are not represented, because they need not switch. `S1`, `S2`, and `S3` are slaves running with `--log-bin` [1181] and without `--log-slave-updates` [1173]. Because updates received by a slave from the master are not logged in the binary log unless `--log-slave-updates` [1173] is specified, the binary log on each slave is empty initially. If for some reason `M` becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick `S1`, all `WC` should be redirected to `S1`, which will log updates to its binary log. `S2` and `S3` should then replicate from `S1`.

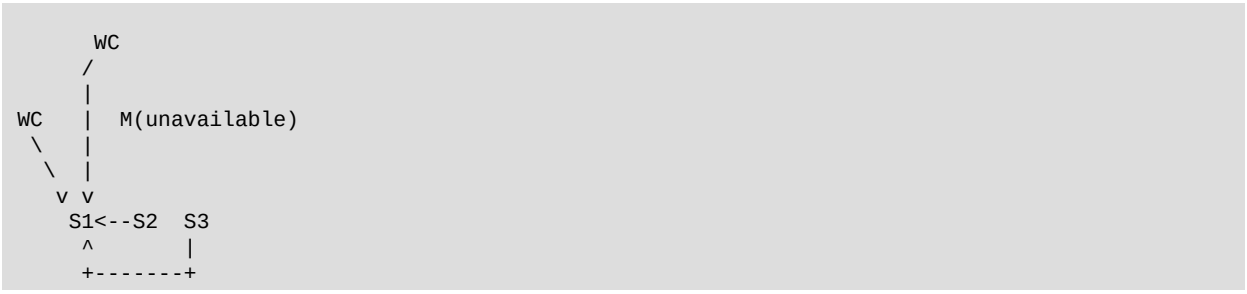
The reason for running the slave without `--log-slave-updates` [1173] is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. Suppose that `S1` has `--log-slave-updates` [1173] enabled. Then it will write updates that it receives from `M` to its own binary log. When `S2` changes from `M` to `S1` as its master, it may receive updates from `S1` that it has already received from `M`

Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave `S1` being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

On the other slaves `S2` and `S3`, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='S1'` (where `'S1'` represents the real host name of `S1`). To use `CHANGE MASTER TO`, add all information about how to connect to `S1` from `S2` or `S3` (*`user`*, *`password`*, *`port`*). In `CHANGE MASTER TO`, there is no need to specify the name of the `S1` binary log file or log position to read from: We know it is the first binary log file and position 4, which are the defaults for `CHANGE MASTER TO`. Finally, use `START SLAVE` on `S2` and `S3`.

Then instruct all `WC` to direct their statements to `S1`. From that point on, all updates statements sent by `WC` to `S1` are written to the binary log of `S1`, which then contains every update statement sent to `S1` since `M` died.

The result is this configuration:



When `M` is up again, you must issue on it the same `CHANGE MASTER TO` as that issued on `S2` and `S3`, so that `M` becomes a slave of `S1` and picks up all the `WC` writes that it missed while it was down. To make `M` a master again (because it is the most powerful machine, for example), use the preceding procedure as if `S1` was unavailable and `M` was to be the new master. During this procedure, do not forget to run

`RESET MASTER` on `M` before making `S1`, `S2`, and `S3` slaves of `M`. Otherwise, they may pick up old `WC` writes from before the point at which `M` became unavailable.

Note that there is no synchronization between the different slaves to a master. Some slaves might be ahead of others. This means that the concept outlined in the previous example might not work. In practice, however, the relay logs of different slaves will most likely not be far behind the master, so it would work, anyway (but there is no guarantee).

Q: How do I prevent GRANT and REVOKE statements from replicating to slave machines?

A: Start the server with the `--replicate-wild-ignore-table=mysql.%` [1178] option.

Q: Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?

A: Yes.

Q: Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

A: Yes.

14.11 Troubleshooting Replication

If you have followed the instructions but your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is nonzero. If binary logging is not enabled, verify that you are running the master with the `--log-bin` [1181] option.
- Verify that the master and slave both were started with the `--server-id` [1167] option and that the ID value is unique on each server.
- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` [1179] prevents the slave threads from starting until you issue a `START SLAVE` statement.
- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See Section 14.3, “Replication Implementation Details”. If the I/O thread state says `Connecting to master`, verify the privileges for the replication user on the master, the master host name, your DNS setup, whether the master is actually running, and whether it is reachable from the slave.
- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in Section 14.7, “Replication Features and Issues”. If it is a bug, see Section 14.12, “How to Report Replication Bugs or Problems”, for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try the following procedure if it is not feasible to do a full database resynchronization by deleting the slave's databases and copying a new snapshot from the master:

1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave's table identical to the master's and run `START SLAVE`.

2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.

3. If you decide that the slave can skip the next statement from the master, issue the following statements:

   ```
   mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N;
   mysql> START SLAVE;
   ```

   The value of *`N`* should be 1 if the next statement from the master does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()` [874]. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or `LAST_INSERT_ID()` [874] is that they take two events in the binary log of the master.

   See also Section 12.5.2.6, “`SET GLOBAL SQL_SLAVE_SKIP_COUNTER` Syntax”.

4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

14.12 How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in Section 1.8, “How to Report Bugs or Problems”. If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*

2. Run the slave with the `--log-slave-updates` [1173] and `--log-bin` [1181] options. These options cause the slave to log the updates that it receives from the master into its own binary logs.

3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:

   - All binary log files from the master
   - All binary log files from the slave
   - The output of `SHOW MASTER STATUS` from the master at the time you discovered the problem

- The output of `SHOW SLAVE STATUS` from the slave at the time you discovered the problem
- Error logs from the master and the slave

4. Use `mysqlbinlog` to examine the binary logs. The following should be helpful to find the problem statement. *`log_file`* and *`log_pos`* are the `Master_Log_File` and `Read_Master_Log_Pos` values from `SHOW SLAVE STATUS`.

```
shell> mysqlbinlog --start-position=log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at Section 1.8, “How to Report Bugs or Problems”.

Chapter 15 MySQL Cluster

Table of Contents

MySQL Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. It uses the `NDBCLUSTER` storage engine to enable running several computers with MySQL servers and other software in a cluster. This storage engine is available and in binary releases from MySQL-Max 4.1.3. Beginning with MySQL 4.1.10a, it is also available in RPMs compatible with most modern Linux distributions. (If you install using RPM files, note that both the `mysql-server` and `mysql-max` RPMs must be installed to have MySQL Cluster capability.)

MySQL Cluster is currently available and supported on a number of platforms, including Linux, Solaris, Mac OS X, and other Unix-style operating systems on a variety of hardware. For exact levels of support available for on specific combinations of operating system versions, operating system distributions, and hardware platforms, please refer to `http://www.mysql.com/support/supportedplatforms/cluster.html`, maintained by the MySQL Support Team on the MySQL web site.

Beginning with MySQL Cluster NDB 7.0, MySQL Cluster is available for testing on Microsoft Windows (but not yet for production use). We are working to make Cluster available on all operating systems supported by MySQL; we will update the information provided here as this work continues. However, we do not plan to make MySQL Cluster available on Microsoft Windows in MySQL 4.1 or any other release series prior to MySQL Cluster NDB 7.0, which is based on MySQL 5.1. For more information, see MySQL Cluster NDB 6.1 - 7.1.

This chapter represents a work in progress, and its contents are subject to revision as MySQL Cluster continues to evolve. Additional information regarding MySQL Cluster can be found on the MySQL Web site at http://www.mysql.com/products/cluster/.

Additional Resources. More information may be found in the following places:

- Answers to some commonly asked questions about Cluster may be found in the Section 15.6, “MySQL 4.1 FAQ: MySQL Cluster”.
- The MySQL Cluster mailing list: http://lists.mysql.com/cluster.
- The MySQL Cluster Forum: http://forums.mysql.com/list.php?25.
- Many MySQL Cluster users and some of the MySQL Cluster developers blog about their experiences with Cluster, and make feeds of these available through PlanetMySQL.
- If you are new to MySQL Cluster, you may find our Developer Zone article How to set up a MySQL Cluster for two servers to be helpful.

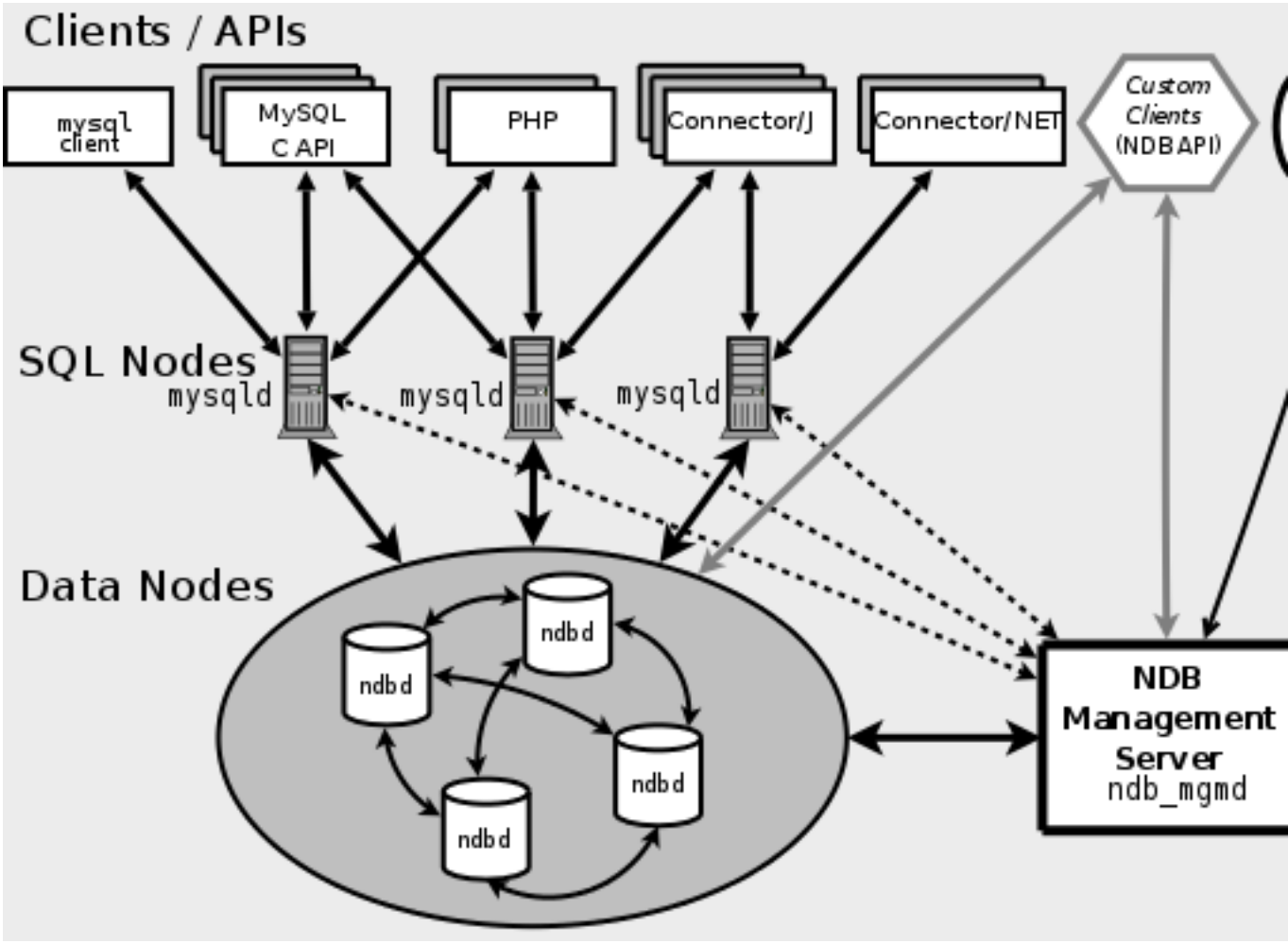
15.1 MySQL Cluster Overview

MySQL Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture enables the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.

MySQL Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network file systems, and SANs is not recommended or supported.

MySQL Cluster integrates the standard MySQL server with an in-memory clustered storage engine called NDB (which stands for "*N*etwork *D*ata*B*ase"). In our documentation, the term NDB refers to the part of the setup that is specific to the storage engine, whereas "MySQL Cluster" refers to the combination of one or more MySQL servers with the NDB storage engine.

A MySQL Cluster consists of a set of computers, known as *hosts*, each running one or more processes. These processes, known as *nodes*, may include MySQL servers (for access to NDB data), data nodes (for storage of the data), one or more management servers, and possibly other specialized data access programs. The relationship of these components in a MySQL Cluster is shown here:



All these programs work together to form a MySQL Cluster (see Section 15.4, "MySQL Cluster Programs". When data is stored by the NDB storage engine, the tables (and table data) are stored in the data nodes. Such tables are directly accessible from all other MySQL servers in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

The data stored in the data nodes for MySQL Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

Individual nodes can be stopped and restarted, and can then rejoin the system (cluster). Rolling restarts (in which all nodes are restarted in turn) are used in making configuration changes and software upgrades

(see Section 15.2.6.1, “Performing a Rolling Restart of a MySQL Cluster”). For more information about data nodes, how they are organized in a MySQL Cluster, and how they handle and store MySQL Cluster data, see Section 15.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”.

Backing up and restoring MySQL Cluster databases can be done using the NDB native functionality found in the MySQL Cluster management client and the `ndb_restore` program included in the MySQL Cluster distribution. For more information, see Section 15.5.3, “Online Backup of MySQL Cluster”, and Section 15.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”. You can also use the standard MySQL functionality provided for this purpose in `mysqldump` and the MySQL server. See Section 4.5.4, “`mysqldump` — A Database Backup Program”, for more information.

MySQL Cluster nodes can use a number of different transport mechanisms for inter-node communications, including TCP/IP using standard 100 Mbps or faster Ethernet hardware. It is also possible to use the high-speed *Scalable Coherent Interface* (SCI) protocol with MySQL Cluster, although this is not required to use MySQL Cluster. SCI requires special hardware and software; see Section 15.3.5, “Using High-Speed Interconnects with MySQL Cluster”, for more about SCI and using it with MySQL Cluster.

15.1.1 MySQL Cluster Core Concepts

`NDBCLUSTER` (also known as `NDB`) is an in-memory storage engine offering high-availability and data-persistence features.

The `NDBCLUSTER` storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. MySQL Cluster's `NDB` storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The “Cluster” portion of MySQL Cluster is configured independently of the MySQL servers. In a MySQL Cluster, each part of the cluster is considered to be a *node*.

Note

In many contexts, the term “node” is used to indicate a computer, but when discussing MySQL Cluster it means a *process*. It is possible to run multiple nodes on a single computer; for a computer on which one or more cluster nodes are being run we use the term *cluster host*.

However, MySQL 4.1 does not support the use of multiple data nodes on a single computer in a production setting. See Section 15.1.4.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

There are three types of cluster nodes, and in a minimal MySQL Cluster configuration, there will be at least three nodes, one of each of these types:

- *Management node* (MGM node): The role of this type of node is to manage the other nodes within the MySQL Cluster, performing such functions as providing configuration data, starting and stopping nodes, running backup, and so forth. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. An MGM node is started with the command `ndb_mgmd`.

- *Data node*: This type of node stores cluster data. There are as many data nodes as there are replicas, times the number of fragments (see Section 15.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”). For example, with two replicas, each having two fragments, you need four data nodes. One replica is sufficient for data storage, but provides no redundancy; therefore, it is recommended to have 2 (or more) replicas to provide redundancy, and thus high availability. A data node is started with the command `ndbd` (see Section 15.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”).

MySQL Cluster tables in MySQL 4.1 are stored completely in memory rather than on disk (this is why we refer to MySQL cluster as an *in-memory* database). In MySQL 5.1, MySQL Cluster NDB 6.X, and later, some MySQL Cluster data can be stored on disk, but we do not expect to backport this functionality to MySQL 4.1; see MySQL Cluster Disk Data Tables, for more information.

- *SQL node*: This is a node that accesses the cluster data. In the case of MySQL Cluster, an SQL node is a traditional MySQL server that uses the `NDBCLUSTER` storage engine. An SQL node is a `mysqld` process started with the `--ndbcluster` [1301] and `--ndb-connectstring` options, which are explained elsewhere in this chapter, possibly with additional MySQL server options as well.

  An SQL node is actually just a specialized type of *API node*, which designates any application which accesses Cluster data. Another example of an API node is the `ndb_restore` utility that is used to restore a cluster backup. It is possible to write such applications using the NDB API. For basic information about the NDB API, see Getting Started with the NDB API.

Important

It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; to benefit from MySQL Cluster's high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, replicas, and partitions in MySQL Cluster, see Section 15.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”.

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. MySQL Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server (MGM node) manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These are of two types:

- **Standard MySQL clients.** MySQL Cluster can be used with existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on. Such client applications send SQL statements to and receive responses from MySQL servers acting as MySQL Cluster SQL nodes in much the same way that they interact with standalone MySQL servers. However, MySQL clients using a MySQL Cluster as a data source can be modified to take advantage of the ability to connect with multiple MySQL servers to achieve load balancing and failover. For example, Java clients using Connector/J 5.0.6 and later can use `jdbc:mysql:loadbalance://` URLs (improved in Connector/J 5.1.7) to achieve load balancing transparently .

- **Management clients.** These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on. Such clients—such as the `ndb_mgm` management client supplied with MySQL Cluster (see Section 15.4.3, “`ndb_mgm` — The MySQL Cluster Management Client”)—are written using the MGM API, a C-language API that communicates directly with one or more MySQL Cluster management servers. For more information, see The MGM API.

Event logs. MySQL Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in Section 15.5.5, “Event Reports Generated in MySQL Cluster”. Event logs are of two types:

- **Cluster log.** Keeps a record of all desired reportable events for the cluster as a whole.
- **Node log.** A separate log which is also kept for each individual node.

Note

Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

Checkpoint. Generally speaking, when data is saved to disk, it is said that a checkpoint has been reached. More specific to Cluster, it is a point in time where all committed transactions are stored on disk. With regard to the `NDB` storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained:

- **Local Checkpoint (LCP).** This is a checkpoint that is specific to a single node; however, LCP's take place for all nodes in the cluster more or less concurrently. An LCP involves saving all of a node's data to disk, and so usually occurs every few minutes. The precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.
- **Global Checkpoint (GCP).** A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.

15.1.2 MySQL Cluster Nodes, Node Groups, Replicas, and Partitions

This section discusses the manner in which MySQL Cluster divides and duplicates data for storage.

Central to an understanding of this topic are the following concepts, listed here with brief definitions:

- **(Data) Node.** An `ndbd` process, which stores a *replica* —that is, a copy of the *partition* (see below) assigned to the node group of which the node is a member.

  Each data node should be located on a separate computer. While it is also possible to host multiple `ndbd` processes on a single computer, such a configuration is not supported.

  It is common for the terms “node” and “data node” to be used interchangeably when referring to an `ndbd` process; where mentioned, management (MGM) nodes (`ndb_mgmd` processes) and SQL nodes (`mysqld` processes) are specified as such in this discussion.

- **Node Group.** A node group consists of one or more nodes, and stores partitions, or sets of *replicas* (see next item).

  The number of node groups in a MySQL Cluster is not directly configurable; it is function of the number of data nodes and of the number of replicas (`NoOfReplicas` configuration parameter), as shown here:

  ```
  [number_of_node_groups] = number_of_data_nodes / NoOfReplicas
  ```

  Thus, a MySQL Cluster with 4 data nodes has 4 node groups if `NoOfReplicas` is set to 1 in the `config.ini` file, 2 node groups if `NoOfReplicas` is set to 2, and 1 node group if `NoOfReplicas` is set to 4. Replicas are discussed later in this section; for more information about `NoOfReplicas`, see Section 15.3.2.5, “Defining MySQL Cluster Data Nodes”.

Note

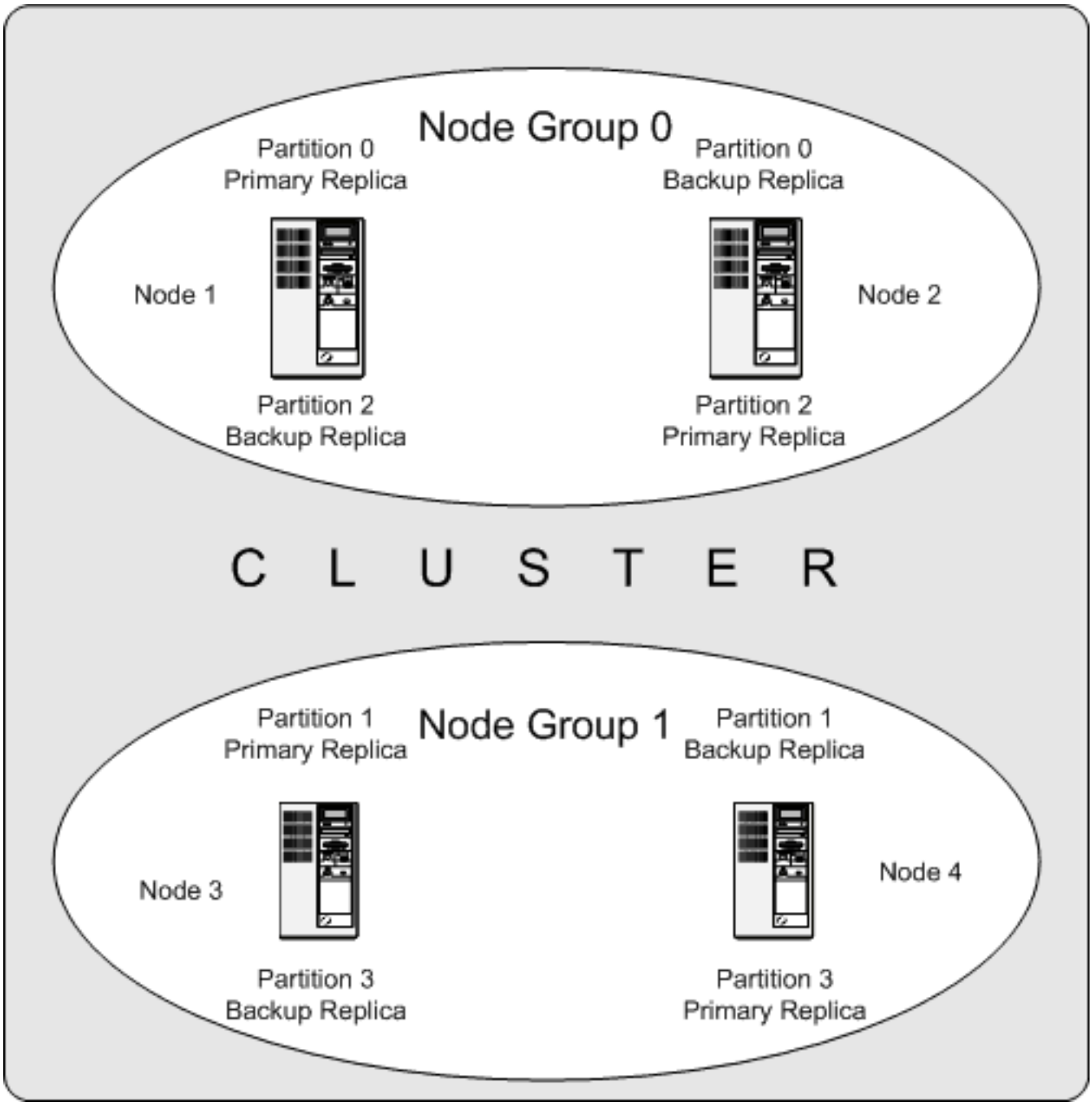
All node groups in a MySQL Cluster must have the same number of data nodes.

- **Partition.** This is a portion of the data stored by the cluster. There are as many cluster partitions as nodes participating in the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one replica) available to the cluster.

  A replica belongs entirely to a single node; a node can (and usually does) store several replicas.

- **Replica.** This is a copy of a cluster partition. Each node in a node group stores a replica. Also sometimes known as a *partition replica*. The number of replicas is equal to the number of nodes per node group.

The following diagram illustrates a MySQL Cluster with four data nodes, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1. Note that only data (`ndbd`) nodes are shown here; although a working cluster requires an `ndb_mgm` process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted in the figure for clarity.

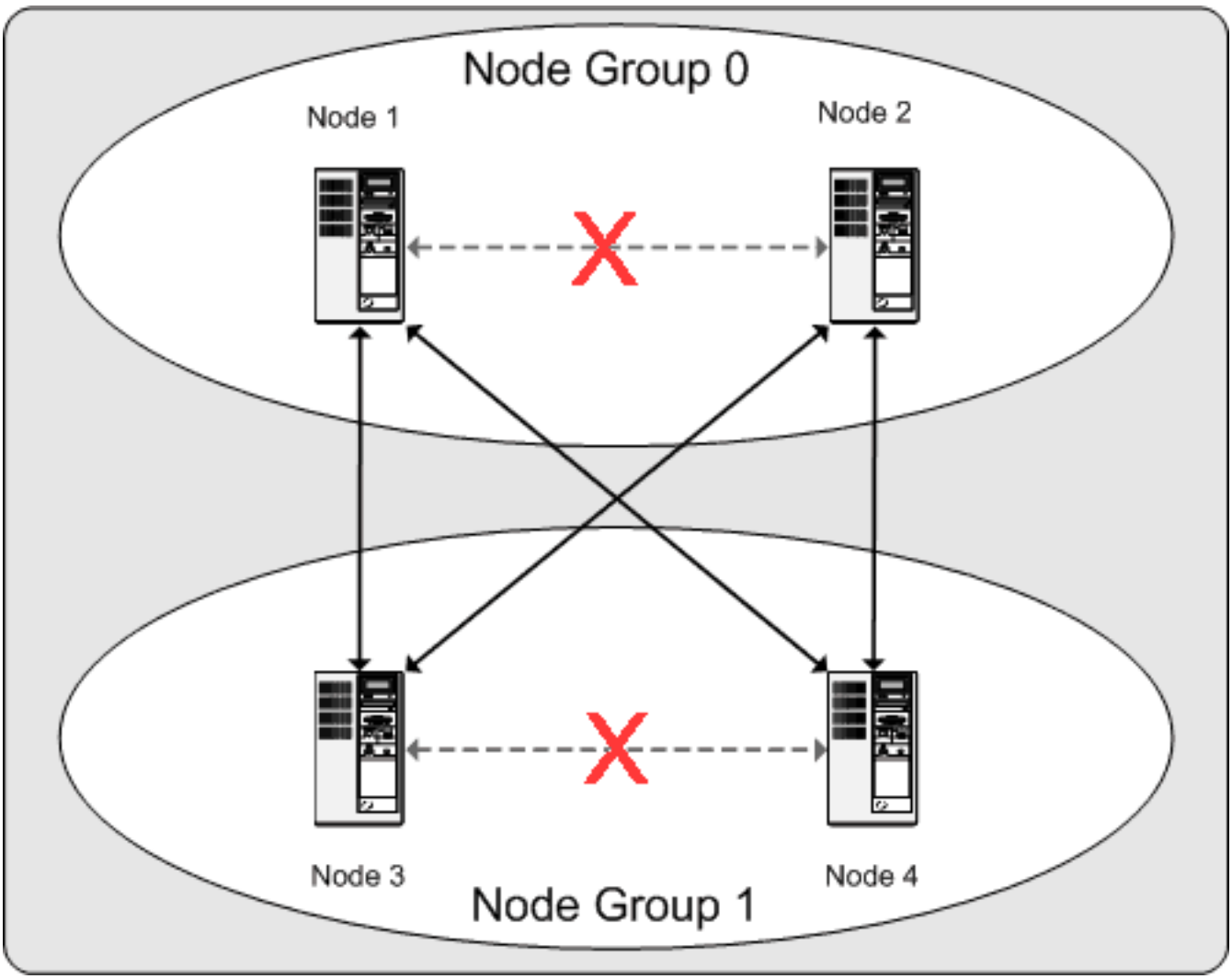


The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored—in multiple copies—on the same node group. Partitions are stored on alternate node groups:

- Partition 0 is stored on node group 0; a *primary replica* (primary copy) is stored on node 1, and a *backup replica* (backup copy of the partition) is stored on node 2.
- Partition 1 is stored on the other node group (node group 1); this partition's primary replica is on node 3, and its backup replica is on node 4.
- Partition 2 is stored on node group 0. However, the placing of its two replicas is reversed from that of Partition 0; for Partition 2, the primary replica is stored on node 2, and the backup on node 1.

- Partition 3 is stored on node group 1, and the placement of its two replicas are reversed from those of partition 1. That is, its primary replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of a MySQL Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.



In this example, where the cluster consists of two node groups of two nodes each, any combination of at least one node in node group 0 and at least one node in node group 1 is sufficient to keep the cluster “alive” (indicated by arrows in the diagram). However, if *both* nodes from *either* node group fail, the remaining two nodes are not sufficient (shown by the arrows marked out with an **X**); in either case, the cluster has lost an entire partition and so can no longer provide access to a complete set of all cluster data.

15.1.3 MySQL Cluster Hardware, Software, and Networking Requirements

One of the strengths of MySQL Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (It is possible to reduce this requirement using Disk Data tables, which were implemented in MySQL 5.1; however, we do not intend to backport this feature to MySQL 4.1.) Naturally, multiple and faster CPUs will enhance performance. Memory requirements for other MySQL Cluster processes are relatively small.

The software requirements for MySQL Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support MySQL Cluster. For supported operating systems, a standard installation should be sufficient. The MySQL software requirements are simple: all that is needed is a production release of MySQL-max 4.1.3 or newer; you must use the `-max` version of MySQL to have MySQL Cluster support. (See Section 5.2, “The `mysqld-max` Extended MySQL Server”.) It is not necessary to compile MySQL yourself merely to be able to use MySQL Cluster. We assume that you are using the server binary appropriate to your platform, available from the MySQL Cluster software downloads page at http://dev.mysql.com/downloads/cluster/.

For communication between nodes, MySQL Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We strongly recommend that a MySQL Cluster be run on its own subnet which is not shared with machines not forming part of the cluster for the following reasons:

- **Security.** Communications between MySQL Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within a MySQL Cluster is to run your MySQL Cluster on a protected network. If you intend to use MySQL Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarized Zone (DMZ) or elsewhere.

  See Section 15.5.9.1, “MySQL Cluster Security and Networking Issues”, for more information.

- **Efficiency.** Setting up a MySQL Cluster on a private or protected network enables the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your MySQL Cluster not only helps protect against unauthorized access to MySQL Cluster data, it also ensures that MySQL Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

It is also possible to use the high-speed Scalable Coherent Interface (SCI) with MySQL Cluster, but this is not a requirement. See Section 15.3.5, “Using High-Speed Interconnects with MySQL Cluster”, for more about this protocol and its use with MySQL Cluster.

15.1.4 Known Limitations of MySQL Cluster

In the sections that follow, we discuss known limitations of MySQL Cluster in MySQL 4.1 as compared with the features available when using the `MyISAM` and `InnoDB` storage engines. Currently, there are no plans to address these in coming releases of MySQL 4.1; however, we will attempt to supply fixes for these issues in subsequent release series. If you check the “Cluster” category in the MySQL bugs database at http://bugs.mysql.com, you can find known bugs which (if marked “4.1”) we intend to correct in upcoming releases of MySQL 4.1.

This information is intended to be complete with respect to the conditions just set forth. You can report any discrepancies that you encounter to the MySQL bugs database using the instructions given in Section 1.8, “How to Report Bugs or Problems”. If we do not plan to fix the problem in MySQL 4.1, we will add it to the list.

15.1.4.1 Noncompliance with SQL Syntax in MySQL Cluster

Some SQL statements relating to certain MySQL features produce errors when used with `NDB` tables, as described in the following list:

- **Temporary tables.** Temporary tables are not supported. Trying either to create a temporary table that uses the `NDB` storage engine or to alter an existing temporary table to use `NDB` fails with the error `Table storage engine 'ndbcluster' does not support the create option 'TEMPORARY'`.

- **Indexes and keys in `NDB` tables.** Keys and indexes on MySQL Cluster tables are subject to the following limitations:

  - **Column width.** Attempting to create an index on an `NDB` table column whose width is greater than 3072 bytes succeeds, but only the first 3072 bytes are actually used for the index. In such cases, a warning `Specified key was too long; max key length is 3072 bytes` is issued, and a `SHOW CREATE TABLE` statement shows the length of the index as 3072.

  - **`TEXT` and `BLOB` columns.** You cannot create indexes on `NDB` table columns that use any of the `TEXT` or `BLOB` data types.

  - **`FULLTEXT` indexes.** The `NDB` storage engine does not support `FULLTEXT` indexes, which are possible for `MyISAM` tables only.

    However, you can create indexes on `VARCHAR` columns of `NDB` tables.

  - **Prefixes.** There are no prefix indexes; only entire columns can be indexed. (The size of an `NDB` column index is always the same as the width of the column in bytes, up to and including 3072 bytes, as described earlier in this section. Also see Section 15.1.4.6, "Unsupported or Missing Features in MySQL Cluster", for additional information.)

  - **`BIT` columns.** A `BIT` column cannot be a primary key, unique key, or index, nor can it be part of a composite primary key, unique key, or index.

  - **`AUTO_INCREMENT` columns.** Like other MySQL storage engines, the `NDB` storage engine can handle a maximum of one `AUTO_INCREMENT` column per table. However, in the case of a Cluster table with no explicit primary key, an `AUTO_INCREMENT` column is automatically defined and used as a "hidden" primary key. For this reason, you cannot define a table that has an explicit `AUTO_INCREMENT` column unless that column is also declared using the `PRIMARY KEY` option. Attempting to create a table with an `AUTO_INCREMENT` column that is not the table's primary key, and using the `NDB` storage engine, fails with an error.

- **MySQL Cluster and geometry data types.** Geometry data types (`WKT` and `WKB`) are supported in `NDB` tables in MySQL 4.1. However, spatial indexes are not supported.

- **Character set support.** Not all charsets and collations are supported. For a list of those that are supported, see http://dev.mysql.com/doc/relnotes/mysql-cluster/4.1/en/mysql-cluster-news-4-1-6.html.

- **Character set directory.** `ndbd` searches only the default path (typically `/usr/local/mysql/share/mysql/charsets`) for character sets. Thus, it is not possible to install MySQL with Cluster support in a different path (in the case of the `.tar.gz` archives, other than `/usr/local/mysql`) if character sets that are not compiled into the MySQL Server need to be used.

15.1.4.2 Limits and Differences of MySQL Cluster from Standard MySQL Limits

In this section, we list limits found in MySQL Cluster that either differ from limits found in, or that are not found in, standard MySQL.

Memory usage and recovery. Memory comsumed when data is inserted into an `NDB` table is not automatically recovered when deleted, as it is with other storage engines. Instead, the following rules hold true:

- A `DELETE` statement on an `NDB` table makes the memory formerly used by the deleted rows available for re-use by inserts on the same table only. However, this memory can be made available for general re-use by performing a rolling restart of the cluster. See Section 15.2.6.1, "Performing a Rolling Restart of a MySQL Cluster".

- A `DROP TABLE` or `TRUNCATE TABLE` operation on an `NDB` table frees the memory that was used by this table for re-use by any `NDB` table, either by the same table or by another `NDB` table.

  Note

  Recall that `TRUNCATE TABLE` drops and re-creates the table. See Section 12.1.10, "`TRUNCATE TABLE` Syntax".

- **Limits imposed by the cluster's configuration.**
  A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in Section 15.3.2, "MySQL Cluster Configuration Files". Most configuration parameters can be upgraded online. These hard limits include:

  - Database memory size and index memory size (`DataMemory` and `IndexMemory`, respectively).

    `DataMemory` is allocated as 32KB pages. As each `DataMemory` page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

    See Section 15.3.2.5, "Defining MySQL Cluster Data Nodes", for further information about `DataMemory` and `IndexMemory`.

  - The maximum number of operations that can be performed per transaction is set using the configuration parameters `MaxNoOfConcurrentOperations` and `MaxNoOfLocalOperations`.

    Note

    Bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.

  - Different limits related to tables and indexes. For example, the maximum number of ordered indexes in the cluster is determined by `MaxNoOfOrderedIndexes`, and the maximum number of ordered inexes per table is 16.

- **Memory usage.** All Cluster table rows are of fixed length. This means (for example) that if a table has one or more `VARCHAR` fields containing only relatively small values, more memory and disk space is required when using the `NDB` storage engine than would be the case for the same table and data using the `MyISAM` engine. (In other words, in the case of a `VARCHAR` column, the column requires the same amount of storage as a `CHAR` column of the same size.)

- **Node and data object maximums.** The following limits apply to numbers of cluster nodes and metadata objects:

  - The maximum number of data nodes is 48.

    A data node must have a node ID in the range of 1-49, inclusive. (Management and API nodes may use any integer in the range of 1-63 inclusive as a node ID.)

  - The total maximum number of nodes in a MySQL Cluster is 63. This number includes all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.

  - The maximum number of metadata objects is limited to 1600, including database tables, system tables, indexes and `BLOB` columns.

15.1.4.3 Limits Relating to Transaction Handling in MySQL Cluster

A number of limitations exist in MySQL Cluster with regard to the handling of transactions. These include the following:

- **Transaction isolation level.** The NDBCLUSTER storage engine supports only the READ COMMITTED [975] transaction isolation level. (InnoDB, for example, supports READ COMMITTED [975], READ UNCOMMITTED [975], REPEATABLE READ [976], and SERIALIZABLE [976].) See Section 15.5.3.4, “MySQL Cluster Backup Troubleshooting”, for information on how this can affect backing up and restoring Cluster databases.)

- **Transactions and BLOB or TEXT columns.** NDBCLUSTER stores only part of a column value that uses any of MySQL's BLOB or TEXT data types in the table visible to MySQL; the remainder of the BLOB or TEXT is stored in a separate internal table that is not accessible to MySQL. This gives rise to two related issues of which you should be aware whenever executing SELECT statements on tables that contain columns of these types:

  1. For any SELECT from a MySQL Cluster table: If the SELECT includes a BLOB or TEXT column, the READ COMMITTED [975] transaction isolation level is converted to a read with read lock. This is done to guarantee consistency.

  2. For any SELECT which uses a primary key lookup or unique key lookup to retrieve any columns that use any of the BLOB or TEXT data types and that is executed within a transaction, a shared read lock is held on the table for the duration of the transaction—that is, until the transaction is either committed or aborted. This does not occur for queries that use index or table scans.

     For example, consider the table t defined by the following CREATE TABLE statement:

     ```
     CREATE TABLE t (
         a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
         b INT NOT NULL,
         c INT NOT NULL,
         d TEXT,
         INDEX i(b),
         UNIQUE KEY u(c)
     ) ENGINE = NDB,
     ```

     Either of the following queries on t causes a shared read lock, because the first query uses a primary key lookup and the second uses a unique key lookup:

     ```
     SELECT * FROM t WHERE a = 1;

     SELECT * FROM t WHERE c = 1;
     ```

     However, none of the four queries shown here causes a shared read lock:

     ```
     SELECT * FROM t WHERE b 1;

     SELECT * FROM t WHERE d = '1';

     SELECT * FROM t;

     SELECT b,c WHERE a = 1;
     ```

     This is because, of these four queries, the first uses an index scan, the second and third use table scans, and the fourth, while using a primary key lookup, does not retrieve the value of any BLOB or TEXT columns.

You can help minimize issues with shared read locks by avoiding queries that use primary key lookups or unique key lookups to retrieve `BLOB` or `TEXT` columns, or, in cases where such queries are not avoidable, by committing transactions as soon as possible afterward.

We are working on overcoming this limitation in a future MySQL Cluster release (see Bug #49190); however, we do not plan to backport any fix for this issue to MySQL 4.1 or MySQL 5.0.

- **Rollbacks.** There are no partial transactions, and no partial rollbacks of transactions. A duplicate key or similar error aborts the entire transaction, and subsequent statements raise `ERROR 1296 (HY000): Got error 4350 'Transaction already aborted' from NDBCLUSTER`. In such cases, you must issue an explicit `ROLLBACK` and retry the entire transaction.

  This behavior differs from that of other transactional storage engines such as `InnoDB` that may roll back individual statements.

- **Transactions and memory usage.**
  As noted elsewhere in this chapter, MySQL Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations. Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behavior of a number of MySQL statements is effected as described in the following list:

  - `TRUNCATE TABLE` is not transactional when used on `NDB` tables. If a `TRUNCATE TABLE` fails to empty the table, then it must be re-run until it is successful.

  - `DELETE FROM` (even with no `WHERE` clause) *is* transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM ... LIMIT ...` statements to "chunk" the delete operation. If your objective is to empty the table, then you may wish to use `TRUNCATE TABLE` instead.

  - **`LOAD DATA` statements.** `LOAD DATA INFILE` is not transactional when used on `NDB` tables. `LOAD DATA FROM MASTER` is not supported in MySQL Cluster.

    Important

    When executing a `LOAD DATA INFILE` statement, the `NDB` engine performs commits at irregular intervals that enable better utilization of the communication network. It is not possible to know ahead of time when such commits take place.

  - **`ALTER TABLE` and transactions.** When copying an `NDB` table as part of an `ALTER TABLE`, the creation of the copy is nontransactional. (In any case, this operation is rolled back when the copy is deleted.)

15.1.4.4 MySQL Cluster Error Handling

Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:

- **Temporary errors.** When first starting a node, it is possible that you may see Error 1204 `Temporary failure, distribution changed` and similar temporary errors.

- **Errors due to node failure.** The stopping or failure of any data node can result in a number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

See also Section 15.1.4.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”.

15.1.4.5 Limits Associated with Database Objects in MySQL Cluster

Some database objects such as tables and indexes have different limitations when using the `NDBCLUSTER` storage engine:

- **Identifiers.** Database names, table names and attribute names cannot be as long in `NDB` tables as when using other table handlers. Attribute names are truncated to 31 characters, and if not unique after truncation give rise to errors. Database names and table names can total a maximum of 122 characters. In other words, the maximum length for an `NDB` table name is 122 characters, less the number of characters in the name of the database of which that table is a part.
- **Table names containing special characters.** `NDB` tables whose names contain characters other than letters, numbers, dashes, and underscores and which are created on one SQL node may not be discovered correctly by other SQL nodes. (Bug #31470)
- **Number of tables and other database objects.** The maximum number of tables in a Cluster database in MySQL 4.1 is limited to 1792. The maximum number of *all* `NDBCLUSTER` database objects in a single MySQL Cluster—including databases, tables, and indexes—is limited to 20320.
- **Attributes per table.** The maximum number of attributes (that is, columns and indexes) per table is limited to 128.
- **Attributes per key.** The maximum number of attributes per key is 32.
- **Row size.** The maximum permitted size of any one row is 8052 bytes. Each `BLOB` or `TEXT` column contributes 256 + 8 = 264 bytes to this total.

15.1.4.6 Unsupported or Missing Features in MySQL Cluster

A number of features supported by other storage engines are not supported for `NDB` tables. Trying to use any of these features in MySQL Cluster does not cause errors in or of itself; however, errors may occur in applications that expects the features to be supported or enforced:

- **Foreign key constraints.** The foreign key construct is ignored, just as it is in `MyISAM` tables.
- **Index prefixes.** Prefixes on indexes are not supported for `NDBCLUSTER` tables. If a prefix is used as part of an index specification in a statement such as `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`, the prefix is ignored.
- **`OPTIMIZE` operations.** `OPTIMIZE` operations are not supported.
- **`LOAD TABLE ... FROM MASTER`.** `LOAD TABLE ... FROM MASTER` is not supported.
- **Savepoints and rollbacks.** Savepoints and rollbacks to savepoints are ignored as in `MyISAM`.
- **Durability of commits.** There are no durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.
- **Replication.** Replication is not supported.

Note

See Section 15.1.4.3, “Limits Relating to Transaction Handling in MySQL Cluster”, for more information relating to limitations on transaction handling in `NDB`.

15.1.4.7 Limitations Relating to Performance in MySQL Cluster

The following performance issues are specific to or especially pronounced in MySQL Cluster:

- **Range scans.** There are query performance issues due to sequential access to the `NDB` storage engine; it is also relatively more expensive to do many range scans than it is with either `MyISAM` or `InnoDB`.
- **Query cache.** The query cache is disabled, since it is not invalidated if an update occurs on a different MySQL server.
- **Reliability of `Records in range`.** The `Records in range` statistic is available but is not completely tested or officially supported. This may result in nonoptimal query plans in some cases. If necessary, you can employ `USE INDEX` or `FORCE INDEX` to alter the execution plan. See Section 12.2.7.2, "Index Hint Syntax", for more information on how to do this.
- **Unique hash indexes.** Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.

15.1.4.8 Issues Exclusive to MySQL Cluster

The following are limitations specific to the `NDBCLUSTER` storage engine:

- **Machine architecture.** The following issues relate to physical architecture of cluster hosts:
  - All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running `mysql` or other clients that may be accessing the cluster's SQL nodes.
  - **Adding and dropping of data nodes.** Online adding or dropping of data nodes is not currently possible. In such cases, the entire cluster must be restarted.
  - **Backup and restore between architectures.** It is also not possible to perform a Cluster backup and restore between different architectures. For example, you cannot back up a cluster running on a big-endian platform and then restore from that backup to a cluster running on a little-endian system. (Bug #19255)
- **Online schema changes.** It is not possible to make online schema changes such as those accomplished using `ALTER TABLE` or `CREATE INDEX`, as the `NDB Cluster` engine does not support autodiscovery of such changes. (However, you can import or create a table that uses a different storage engine, and then convert it to `NDB` using `ALTER TABLE tbl_name ENGINE=NDBCLUSTER`. In such a case, you must issue a `FLUSH TABLES` statement to force the cluster to pick up the change.)
- **Binary logging.**
  MySQL Cluster has the following limitations or restrictions with regard to binary logging:
  - `sql_log_bin` [429] has no effect on data operations; however, it is supported for schema operations.
  - MySQL Cluster cannot produce a binlog for tables having `BLOB` columns but no primary key.
  - Only the following schema operations are logged in a cluster binlog which is *not* on the `mysqld` executing the statement:
    - `CREATE TABLE`

- `ALTER TABLE`
- `DROP TABLE`
- `CREATE DATABASE` / `CREATE SCHEMA`
- `DROP DATABASE` / `DROP SCHEMA`

See also Section 15.1.4.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

15.1.4.9 Limitations Relating to Multiple MySQL Cluster Nodes

Multiple SQL nodes.
The following are issues relating to the use of multiple MySQL servers as MySQL Cluster SQL nodes, and are specific to the `NDBCLUSTER` storage engine:

- **No distributed table locks.** A `LOCK TABLES` works only for the SQL node on which the lock is issued; no other SQL node in the cluster “sees” this lock. This is also true for a lock issued by any statement that locks tables as part of its operations. (See next item for an example.)
- **`ALTER TABLE` operations.** `ALTER TABLE` is not fully locking when running multiple MySQL servers (SQL nodes). (As discussed in the previous item, MySQL Cluster does not support distributed table locks.)
- **Replication.** MySQL replication will not work correctly if updates are done on multiple MySQL servers. However, if the database partitioning scheme is done at the application level and no transactions take place across these partitions, replication can be made to work.
- **Database autodiscovery.** Autodiscovery of databases is not supported for multiple MySQL servers accessing the same MySQL Cluster. However, autodiscovery of tables is supported in such cases. What this means is that after a database named *`db_name`* is created or imported using one MySQL server, you should issue a `CREATE DATABASE` *`db_name`* statement on each additional MySQL server that accesses the same MySQL Cluster. (As of MySQL 5.0.2, you may also use `CREATE SCHEMA` *`db_name`*.) Once this has been done for a given MySQL server, that server should be able to detect the database tables without error.
- **DDL operations.** DDL operations are not node failure safe. If a node fails while trying to perform one of these (such as `CREATE TABLE` or `ALTER TABLE`), the data dictionary is locked and no further DDL statements can be executed without restarting the cluster.

Multiple management nodes.
When using multiple management servers:

- You must give nodes explicit IDs in connectstrings because automatic allocation of node IDs does not work across multiple management servers.

  In addition, all API nodes (including MySQL servers acting as SQL nodes), should list all management servers using the same order in their connectstrings.
- You must take extreme care to have the same configurations for all management servers. No special checks for this are performed by the cluster.
- Prior to MySQL 4.1.15, all data nodes had to be restarted after bringing up the cluster for the management nodes to be able to see one another.

  (See Bug #12307 and Bug #13070 for more information.)

Multiple data node processes. While it is possible to run multiple cluster processes concurrently on a single host, it is not always advisable to do so for reasons of performance and high availability, as well as other considerations. In particular, in MySQL 4.1, we do not support for production use any MySQL Cluster deployment in which more than one `ndbd` process is run on a single physical machine.

Note

We may support multiple data nodes per host in a future MySQL release, following additional testing. However, in MySQL 4.1, such configurations can be considered experimental only.

Multiple network addresses. Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.

Note

It is possible to use multiple network hardware *interfaces* (such as Ethernet cards) for a single data node, but these must be bound to the same address. This also means that it not possible to use more than one `[tcp]` section per connection in the `config.ini` file. See Section 15.3.2.7, “MySQL Cluster TCP/IP Connections”, for more information.

15.2 MySQL Cluster Multi-Computer How-To

This section is a “How-To” that describes the basics for how to plan, install, configure, and run a MySQL Cluster. Whereas the examples in Section 15.3, “MySQL Cluster Configuration” provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures outlined here should be a usable MySQL Cluster which meets the *minimum* requirements for availability and safeguarding of data.

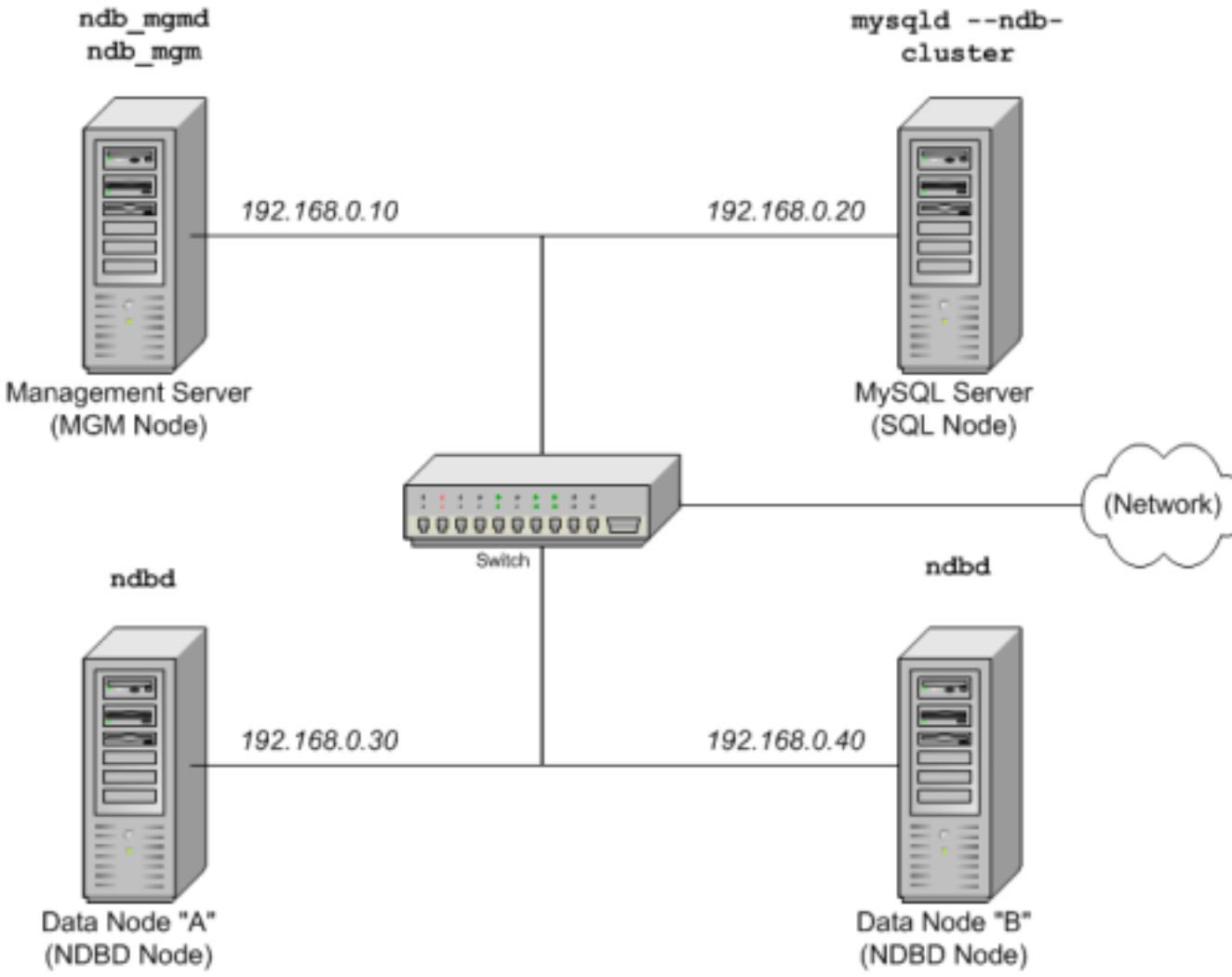
This section covers hardware and software requirements; networking issues; installation of MySQL Cluster; configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

Basic assumptions. This *How-To* makes the following assumptions:

1. The cluster is to be set up with four nodes, each on a separate host, and each with a fixed network address on a typical Ethernet network as shown here:

Node	IP Address
Management (MGMD) node	192.168.0.10
MySQL server (SQL) node	192.168.0.20
Data (NDBD) node "A"	192.168.0.30
Data (NDBD) node "B"	192.168.0.40

   This may be made clearer in the following diagram:



In the interest of simplicity (and reliability), this *How-To* uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use host names in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `/etc/hosts` file or your operating system's equivalent for providing a means to do host lookup if such is available.

Note

A common problem when trying to use host names for Cluster nodes arises because of the way in which some operating systems (including some Linux distributions) set up the system's own host name in the `/etc/hosts` during installation. Consider two machines with the host names `ndb1` and `ndb2`, both in the `cluster` network domain. Red Hat Linux (including some derivatives such as CentOS and Fedora) places the following entries in these machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
127.0.0.1   ndb1.cluster ndb1 localhost.localdomain localhost
```

```
#  ndb2 /etc/hosts:
```

```
127.0.0.1   ndb2.cluster ndb2 localhost.localdomain localhost
```

SUSE Linux (including OpenSUSE) places these entries in the machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb1.cluster ndb1
```

```
#  ndb2 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb2.cluster ndb2
```

In both instances, `ndb1` routes `ndb1.cluster` to a loopback IP address, but gets a public IP address from DNS for `ndb2.cluster`, while `ndb2` routes `ndb2.cluster` to a loopback address and obtains a public address for `ndb1.cluster`. The result is that each data node connects to the management server, but cannot tell when any other data nodes have connected, and so the data nodes appear to hang while starting.

You should also be aware that you cannot mix `localhost` and other host names or IP addresses in `config.ini`. For these reasons, the solution in such cases (other than to use IP addresses for *all* `config.ini HostName` entries) is to remove the fully qualified host names from `/etc/hosts` and use these in `config.ini` for all cluster hosts.

2. Each host in our scenario is an Intel-based desktop PC running a supported operating system installed to disk in a standard configuration, and running no unnecessary services. The core operating system with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the file systems on all hosts are set up identically. In the event that they are not, you should adapt these instructions accordingly.

3. Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected through a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughout. That is, all four machines in the cluster should have 100 Mbps cards *or* all four machines should have 1 Gbps cards.) MySQL Cluster works in a 100 Mbps network; however, gigabit Ethernet provides better performance.

   Note that MySQL Cluster is *not* intended for use in a network for which throughput is less than 100 Mbps or which experiences a high degree of latency. For this reason (among others), attempting to run a MySQL Cluster over a wide area network such as the Internet is not likely to be successful, and is not supported in production.

4. For our sample data, we use the `world` database which is available for download from the MySQL Web site (see http://dev.mysql.com/doc/index-other.html). We assume that each machine has sufficient memory for running the operating system, host NDB process, and (on the data nodes) storing the database.

Although we refer to a Linux operating system in this How-To, the instructions and procedures that we provide here should be easily adaptable to other supported operating systems. We also assume that you already know how to perform a minimal installation and configuration of the operating system with networking capability, or that you are able to obtain assistance in this elsewhere if needed.

For information about MySQL Cluster hardware, software, and networking requirements, see Section 15.1.3, “MySQL Cluster Hardware, Software, and Networking Requirements”.

15.2.1 MySQL Cluster Multi-Computer Installation

Each MySQL Cluster host computer running an SQL node must have installed on it a MySQL binary. For management nodes and data nodes, it is not necessary to install the MySQL server binary, but management nodes require the management server daemon (`ndb_mgmd`) and data nodes require the data node daemon (`ndbd`). It is also a good idea to install the management client (`ndb_mgm`) on the management server host. This section covers the steps necessary to install the correct binaries for each type of Cluster node.

Oracle provides precompiled binaries that support MySQL Cluster, and there is generally no need to compile these yourself. However, we also include information relating to installing a MySQL Cluster after building MySQL from source. For setting up a cluster using MySQL's binaries, the first step in the installation process for each cluster host is to download the file `mysql-max-4.1.25-pc-linux-gnu-i686.tar.gz` from the MySQL downloads area. We assume that you have placed it in each machine's `/var/tmp` directory. (If you do require a custom binary, see Section 2.9.2, “Installing MySQL from a Development Source Tree”.)

RPMs are also available for both 32-bit and 64-bit Linux platforms. For a MySQL Cluster, four RPMs are required:

- The **Server** RPM (for example, `MySQL-server-4.1.25-0.glibc23.i386.rpm`), which supplies the core files needed to run a MySQL Server.
- The **Server/Max** RPM (for example, `MySQL-Max-4.1.25-0.glibc23.i386.rpm`), which provides a MySQL Server binary with clustering support.
- The **NDB Cluster - Storage engine** RPM (for example, `MySQL-ndb-storage-4.1.25-0.glibc23.i386.rpm`), which supplies the MySQL Cluster data node binary (`ndbd`).
- The **NDB Cluster - Storage engine management RPM** (for example, `MySQL-ndb-management-4.1.25-0.glibc23.i386.rpm`), which provides the MySQL Cluster management server binary (`ndb_mgmd`).

In addition, you should also obtain the **NDB Cluster - Storage engine basic tools** RPM (for example, `MySQL-ndb-tools-4.1.25-0.glibc23.i386.rpm`), which supplies several useful applications for working with a MySQL Cluster. The most important of these is the MySQL Cluster management client (`ndb_mgm`). The **NDB Cluster - Storage engine extra tools** RPM (for example, `MySQL-ndb-extra-4.1.25-0.glibc23.i386.rpm`) contains some additional testing and monitoring programs, but is not required to install a MySQL Cluster. (For more information about these additional programs, see Section 15.4, “MySQL Cluster Programs”.)

The MySQL version number in the RPM file names (shown here as `4.1.25`) can vary according to the version which you are actually using. *It is very important that all of the Cluster RPMs to be installed have the same MySQL version number*. The `glibc` version number (if present—shown here as `glibc23`), and architecture designation (shown here as `i386`) should be appropriate to the machine on which the RPM is to be installed.

See Section 2.4, “Installing MySQL from RPM Packages on Linux”, for general information about installing MySQL using RPMs supplied by Oracle.

After installing from RPM, you still need to configure the cluster as discussed in Section 15.2.2, “MySQL Cluster Multi-Computer Configuration”.

Note

After completing the installation, do not yet start any of the binaries. We show you how to do so following the configuration of all nodes.

Data and SQL Node Installation: `.tar.gz` Binary. On each of the machines designated to host data or SQL nodes, perform the following steps as the system `root` user:

1. Check your `/etc/passwd` and `/etc/group` files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a `mysql` group and `mysql` user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new `mysql` user group, and then add a `mysql` user to this group:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symlink to the `mysql-max` directory named `mysql`. Note that the actual file and directory names will vary according to the MySQL version number.

   ```
   shell> cd /var/tmp
   shell> tar -C /usr/local -xzvf mysql-max-4.1.25-pc-linux-gnu-i686.tar.gz
   shell> ln -s /usr/local/mysql-max-4.1.25-pc-linux-gnu-i686 /usr/local/mysql
   ```

3. Change location to the `mysql` directory and run the supplied script for creating the system databases:

   ```
   shell> cd mysql
   shell> scripts/mysql_install_db --user=mysql
   ```

4. Set the necessary permissions for the MySQL server and data directories:

   ```
   shell> chown -R root .
   shell> chown -R mysql data
   shell> chgrp -R mysql .
   ```

   Note that the data directory on each machine hosting a data node is `/usr/local/mysql/data`. This piece of information is essential when configuring the management node. (See Section 15.2.2, “MySQL Cluster Multi-Computer Configuration”.)

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

   ```
   shell> cp support-files/mysql.server /etc/rc.d/init.d/
   shell> chmod +x /etc/rc.d/init.d/mysql.server
   shell> chkconfig --add mysql.server
   ```

   (The startup scripts directory may vary depending on your operating system and version—for example, in some Linux distributions, it is `/etc/init.d`.)

   Here we use Red Hat's `chkconfig` for creating links to the startup scripts; use whatever means is appropriate for this purpose on your operating system and distribution, such as `update-rc.d` on Debian.

Remember that the preceding steps must be performed separately on each machine where an SQL node is to reside.

SQL node installation: RPM files. On each machine to be used for hosting a cluster SQL node, install the **MySQL Max** RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL web site:

```
shell> rpm -Uhv MySQL-server-4.1.25-0.glibc23.i386.rpm
shell> rpm -Uhv MySQL-Max-4.1.25-0.glibc23.i386.rpm
```

This installs the MySQL Max server binary (`mysqld-max`) in the `/usr/sbin` directory, as well as all needed MySQL Server support files. It also installs the `mysql.server` and `mysqld_safe` startup scripts in `/usr/share/mysql` and `/usr/bin`, respectively. The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

SQL node installation: building from source. If you compile MySQL with clustering support (for example, by using the `BUILD/compile-`*`platform_name`*`-max` script appropriate to your platform), and perform the default installation (using `make install` as the root user), `mysqld` is placed in `/usr/local/mysql/bin`. Follow the steps given in Section 2.9, “Installing MySQL from Source” to make `mysqld` ready for use. If you want to run multiple SQL nodes, you can use a copy of the same `mysqld` executable and its associated support files on several machines. The easiest way to do this is to copy the entire `/usr/local/mysql` directory and all directories and files contained within it to the other SQL node host or hosts, then repeat the steps from Section 2.9, “Installing MySQL from Source” on each machine. If you configure the build with a nondefault `--prefix` [95], you need to adjust the directory accordingly.

Data node installation: RPM Files. On a computer that is to host a cluster data node it is necessary to install only the **NDB Cluster - Storage engine** RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL web site:

```
shell> rpm -Uhv MySQL-ndb-storage-4.1.25-0.glibc23.i386.rpm
```

The previous command installs the MySQL Cluster data node binary (`ndbd`) in the `/usr/sbin` directory.

Data node installation: building from source. The only executable required on a data node host is `ndbd` (`mysqld`, for example, does not have to be present on the host machine). By default when doing a source build, this file is placed in the directory `/usr/local/mysql/libexec`. For installing on multiple data node hosts, only `ndbd` need be copied to the other host machine or machines. (This assumes that all data node hosts use the same architecture and operating system; otherwise you may need to compile separately for each different platform.) `ndbd` need not be in any particular location on the host's file system, as long as the location is known.

Management node installation: `.tar.gz` binary. Installation of the management node does not require the `mysqld` binary. Only the MySQL Cluster management server (`ndb_mgmd`) is required; you most likely want to install the management client (`ndb_mgm`) as well. Both of these binaries also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root` (that is, after using `sudo`, `su root`, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install `ndb_mgmd` and `ndb_mgm` on the Cluster management node host:

1. Change location to the `/var/tmp` directory, and extract the `ndb_mgm` and `ndb_mgmd` from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-4.1.25-pc-linux-gnu-i686.tar.gz
shell> cd mysql-4.1.25-pc-linux-gnu-i686
shell> cp bin/ndb_mgm* /usr/local/bin
```

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

Management node installation: RPM file. To install the MySQL Cluster management server, it is necessary only to use the **NDB Cluster - Storage engine management** RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the **Storage engine management** RPM downloaded from the MySQL web site):

```
shell> rpm -Uhv MySQL-ndb-management-4.1.25-0.glibc23.i386.rpm
```

This installs the management server binary (`ndb_mgmd`) to the `/usr/sbin` directory.

You should also install the `NDB` management client, which is supplied by the **Storage engine basic tools** RPM. Copy this RPM to the same computer as the management node, and then install it by running the following command as the system root user (again, replace the name shown for the RPM as necessary to match that of the **Storage engine basic tools** RPM downloaded from the MySQL web site):

```
shell> rpm -Uhv MySQL-ndb-tools-4.1.25-0.glibc23.i386.rpm
```

The **Storage engine basic tools** RPM installs the MySQL Cluster management client (`ndb_mgm`) to the `/usr/bin` directory.

Management node installation: building from source. When building from source and running the default `make install`, the management server binary (`ndb_mgmd`) is placed in `/usr/local/mysql/libexec`, while the management client binary (`ndb_mgm`) can be found in `/usr/local/mysql/bin`. Only `ndb_mgmd` is required to be present on a management node host; however, it is also a good idea to have `ndb_mgm` present on the same host machine. Neither of these executables requires a specific location on the host machine's file system.

In Section 15.2.2, “MySQL Cluster Multi-Computer Configuration”, we create configuration files for all of the nodes in our example Cluster.

15.2.2 MySQL Cluster Multi-Computer Configuration

For our four-node, four-host MySQL Cluster, it is necessary to write four configuration files, one per node host.

- Each data node or SQL node requires a `my.cnf` file that provides two pieces of information: a *connectstring* that tells the node where to find the management node, and a line telling the MySQL server on this host (the machine hosting the data node) to enable the `NDBCLUSTER` storage engine.

  For more information on connectstrings, see Section 15.3.2.2, “The MySQL Cluster Connectstring”.

- The management node needs a `config.ini` file telling it how many replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the Storage and SQL Nodes

The `my.cnf` file needed for the data nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```

Note

We show `vi` being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, `my.cnf` should look like this:

```
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine
ndb-connectstring=192.168.0.10  # location of management server

[mysql_cluster]
# Options for ndbd process:
ndb-connectstring=192.168.0.10  # location of management server
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important

Once you have started a `mysqld` process with the `NDBCLUSTER` and `ndb-connectstring` parameters in the `[mysqld]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design*.

Configuring the management node. The first step in configuring the management node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as `root`):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2    # Number of replicas
DataMemory=80M    # How much memory to allocate for data storage
IndexMemory=18M   # How much memory to allocate for index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes up
                  # only about 500KB, this should be more than enough for
```

```
                # this example Cluster setup.

[tcp default]
# TCP/IP options:
portnumber=2202   # This the default; however, you can use any
                  # port that is free for all the hosts in the cluster
                  # Note: It is recommended beginning with MySQL 5.0 that
                  # you do not specify the portnumber at all and simply allow
                  # the default value to be used instead

[ndb_mgmd]
# Management process options:
hostname=192.168.0.10           # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster  # Directory for MGM node log files

[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
hostname=192.168.0.30           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data files

[ndbd]
# Options for data node "B":
hostname=192.168.0.40           # Hostname or IP address
datadir=/usr/local/mysql/data   # Directory for this data node's data files

[mysqld]
# SQL node options:
hostname=192.168.0.20           # Hostname or IP address
                                # (additional mysqld connections can be
                                # specified for this node for various
                                # purposes such as running ndb_restore)
```

Note

The `world` database can be downloaded from http://dev.mysql.com/doc/, where it can be found listed under “Examples”.

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in Section 15.2.3, “Initial Startup of MySQL Cluster”.

For more detailed information about the available MySQL Cluster configuration parameters and their uses, see Section 15.3.2, “MySQL Cluster Configuration Files”, and Section 15.3, “MySQL Cluster Configuration”. For configuration of MySQL Cluster as relates to making backups, see Section 15.5.3.3, “Configuration for MySQL Cluster Backups”.

Note

The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. In MySQL 4.1, ports for data nodes are allocated sequentially beginning with port 2202 and these ports must be available for the cluster to use.

15.2.3 Initial Startup of MySQL Cluster

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the management node process:

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

Note

`ndb_mgmd` must be told where to find its configuration file, using the `-f` or `--config-file` option. (See Section 15.4.2, "`ndb_mgmd` — The MySQL Cluster Management Server Daemon", for details.)

For additional options which can be used with `ndb_mgmd`, see Section 15.4.20, "Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs".

2. On each of the data node hosts, run this command to start the `ndbd` process:

```
shell> ndbd
```

3. If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node. As mention previously, you must install the **Server / Max** RPM *in addition to* the **Server** RPM to obtain and run the `mysqld-max` server binary.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the `ndb_mgm` management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @192.168.0.30  (Version: 4.1.25, Nodegroup: 0, Master)
id=3    @192.168.0.40  (Version: 4.1.25, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @192.168.0.10  (Version: 4.1.25)

[mysqld(API)]   1 node(s)
id=4    @192.168.0.20  (Version: 4.1.25)
```

The SQL node is referenced here as `[mysqld(API)]`, which reflects the fact that the `mysqld` process is acting as a MySQL Cluster API node.

Note

The IP address shown for a given MySQL Cluster SQL or other API node in the output of `SHOW` is the address used by the SQL or API node to connect to the cluster data nodes, and not to any management node.

You should now be ready to work with databases, tables, and data in MySQL Cluster. See Section 15.2.4, "Loading Sample Data into MySQL Cluster and Performing Queries", for a brief discussion.

15.2.4 Loading Sample Data into MySQL Cluster and Performing Queries

Working with data in MySQL Cluster is not much different from doing so in MySQL without Cluster. There are two points to keep in mind:

- For a table to be replicated in the cluster, it must use the `NDBCLUSTER` storage engine. To specify this, use the `ENGINE=NDBCLUSTER` or `ENGINE=NDB` option when creating the table:

  ```
  CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
  ```

  Alternatively, for an existing table that uses a different storage engine, use `ALTER TABLE` to change the table to use `NDBCLUSTER`:

  ```
  ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
  ```

- Each `NDBCLUSTER` table *must* have a primary key. If no primary key is defined by the user when a table is created, the `NDBCLUSTER` storage engine automatically generates a hidden one.

Note

This hidden key takes up space just as does any other table index. It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of `mysqldump`, you can open the SQL script in a text editor and add the `ENGINE` option to any table creation statements, or replace any existing `ENGINE` (or `TYPE`) options. Suppose that you have the `world` sample database on another MySQL server that does not support MySQL Cluster, and you want to export the `City` table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting `city_table.sql` file will contain this table creation statement (and the `INSERT` statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

You need to make sure that MySQL uses the `NDBCLUSTER` storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition *before* importing it into the Cluster database. Using the `City` table as an example, modify the `ENGINE` option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
```

```
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of `TYPE=engine_name` or `ENGINE=engine_name` with `ENGINE=NDBCLUSTER`. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use `ALTER TABLE` to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named `world` on the SQL node of the cluster, you can then use the `mysql` command-line client to read `city_table.sql`, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address `192.168.0.20`).

To create a copy of the entire `world` database on the SQL node, use `mysqldump` on the noncluster server to export the database to a file named `world.sql`; for example, in the `/tmp` directory. Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

It is important to note that `NDBCLUSTER` in MySQL 4.1 does not support autodiscovery of databases. (See Section 15.1.4, “Known Limitations of MySQL Cluster”.) This means that, once the `world` database and its tables have been created on one data node, you need to issue the `CREATE DATABASE world` statement followed by `FLUSH TABLES` on each SQL node in the cluster. This causes the node to recognize the database and read its table definitions.

Running `SELECT` queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the `root` password at the `Enter password:` prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.25

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

We simply use the MySQL server's `root` account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong `root` password. For more information, see Section 2.10.3, “Securing the Initial MySQL Accounts”.

It is worth taking into account that Cluster nodes do *not* make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the `root` account) effects only applications that access the SQL node, not interaction between nodes. See Section 15.5.9.2, “MySQL Cluster and MySQL Privileges”, for more information.

If you did not modify the `ENGINE` clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a `SELECT` query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Bombay    |   10500000 |
| Seoul     |    9981619 |
| São Paulo |    9968485 |
| Shanghai  |    9696300 |
| Jakarta   |    9604900 |
+-----------+------------+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

Applications that use MySQL can employ standard APIs to access `NDB` tables. It is important to remember that your application must access the SQL node, and not the management or data nodes. This brief example shows how we might execute the `SELECT` statement just shown by using the PHP 5.X `mysqli` extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', 'root_password', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";
```

```
  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing ="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
    # then display the results...
    while($row = $result->fetch_object())
      printf("<tr>\n  <td align=\"center\">%s</td><td>%d</td>\n</tr>\n",
              $row->Name, $row->Population);
?>
  </tbody
</table>
<?
  # ...and verify the number of rows that were retrieved
    printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();

  # free the result set and the mysqli connection object
  $result->close();
  $link->close();
?>
</body>
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

15.2.5 Safe Shutdown and Restart of MySQL Cluster

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

```
shell> ndb_mgm -e shutdown
```

The `-e` option here is used to pass a command to the `ndb_mgm` client from the shell. (See Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”, for more information about this option.) The command causes the `ndb_mgm`, `ndb_mgmd`, and any `ndbd` processes to terminate gracefully. Any SQL nodes can be terminated using `mysqladmin shutdown` and other means.

To restart the cluster, run these commands:

- On the management host (`192.168.0.10` in our example setup):

  ```
  shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
  ```

- On each of the data node hosts (`192.168.0.30` and `192.168.0.40`):

  ```
  shell> ndbd
  ```

- On the SQL host (192.168.0.20):

```
shell> mysqld_safe &
```

In a production setting, it is usually not desirable to shut down the cluster completely. In many cases, even when making configuration changes, or performing upgrades to the cluster hardware or software (or both), which require shutting down individual host machines, it is possible to do so without shutting down the cluster as a whole by performing a *rolling restart* of the cluster. For more information about doing this, see Section 15.2.6.1, “Performing a Rolling Restart of a MySQL Cluster”.

15.2.6 Upgrading and Downgrading MySQL Cluster

This portion of the MySQL Cluster chapter covers upgrading and downgrading a MySQL Cluster from one MySQL release to another. It discusses different types of Cluster upgrades and downgrades, and provides a Cluster upgrade/downgrade compatibility matrix (see Section 15.2.6.2, “MySQL Cluster 4.1 Upgrade and Downgrade Compatibility”). You are expected already to be familiar with installing and configuring a MySQL Cluster prior to attempting an upgrade or downgrade. See Section 15.3, “MySQL Cluster Configuration”.

This section remains in development, and continues to be updated and expanded.

15.2.6.1 Performing a Rolling Restart of a MySQL Cluster

This section discusses how to perform a *rolling restart* of a MySQL Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable:

- **Cluster configuration change.**
  To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.

- **Cluster software upgrade/downgrade.** To upgrade the cluster to a newer version of the MySQL Cluster software (or to downgrade it to an older version). This is usually referred to as a “rolling upgrade” (or “rolling downgrade”, when reverting to an older version of MySQL Cluster).

- **Change on node host.** To make changes in the hardware or operating system on which one or more cluster nodes are running.

- **Cluster reset.**
  To reset the cluster because it has reached an undesirable state. In such cases it is often desirable to reload the data and metadata of one or more data nodes. This can be done 1 of 3 ways:

  - Start each data node process (ndbd, or possibly ndbmtd in MySQL Cluster NDB 7.0 and later) with the --initial [1309] option, which forces the data node to clear its filesystem and reload data and metadata from the other data nodes.

  - Create a backup using the ndb_mgm client BACKUP command prior to performing the restart. Following the upgrade, restore the node or nodes using ndb_restore.

    See Section 15.5.3, “Online Backup of MySQL Cluster”, and Section 15.4.14, “ndb_restore — Restore a MySQL Cluster Backup”, for more information.

  - Use `mysqldump` (see Section 4.5.4, “`mysqldump` — A Database Backup Program”) to create a backup prior to the upgrade; afterward, restore the dump using `LOAD DATA INFILE`.

- **Freeing of resources.**
  To permit memory allocated to a table by successive `INSERT` and `DELETE` operations to be freed for re-use by other MySQL Cluster tables.

The process for performing a rolling restart may be generalized as follows:

1. Stop all cluster management nodes (`ndb_mgmd` processes), reconfigure them, then restart them.

2. Stop, reconfigure, then restart each cluster data node (`ndbd` process) in turn.

3. Stop, reconfigure, then restart each cluster SQL node (`mysqld` process) in turn.

The specifics for implementing a particular rolling upgrade depend upon the actual changes being made. A more detailed view of the process is presented here:

<table>
<tr><th colspan="5">RESTART TYPE:</th></tr>
<tr><th>Cluster Configuration Change</th><th>Cluster Software Upgrade or Downgrade</th><th>Change on Node Host</th><th colspan="2">Cluster Reset</th></tr>
<tr><td colspan="5">A. Management node (<code>ndb_mgmd</code>) processes...</td></tr>
<tr><td rowspan="2">1. Stop all <code>ndb_mgmd</code> processes

2. Make changes in global configuration file(s)

3. Start all <code>ndb_mgmd</code> processes</td><td rowspan="2">1. Stop all <code>ndb_mgmd</code> processes

2. Replace each <code>ndb_mgmd</code> binary with new version

3. Start <code>ndb_mgmd</code> processes</td><td rowspan="2">1. Stop all <code>ndb_mgmd</code> processes

2. Make desired changes in hardware, operating system, or both

3. Start all <code>ndb_mgmd</code> processes</td><td colspan="2">(OR)</td></tr>
<tr><td>1. Stop all <code>ndb_mgmd</code> processes

2. Start all <code>ndb_mgmd</code> processes</td><td>Restart all <code>ndb_mgmd</code> processes (optional)</td></tr>
<tr><td colspan="5">B. For each data node (<code>ndbd</code>) process...</td></tr>
<tr><td colspan="2">(OR)</td><td rowspan="2">1. Stop <code>ndbd</code>

2. Replace <code>ndbd</code> binary with new version

3. Start <code>ndbd</code></td><td rowspan="2">1. Stop <code>ndbd</code>

2. Make desired changes in hardware, operating system, or both

3. Start <code>ndbd</code></td><td colspan="2">(OR)</td></tr>
<tr><td>1. Stop <code>ndbd</code>

2. Start <code>ndbd</code></td><td>Restart <code>ndbd</code></td><td>1. Stop <code>ndbd</code>

2. Start <code>ndbd</code></td><td>Restart <code>ndbd</code></td></tr>
<tr><td colspan="5">C. For each SQL node (<code>mysqld</code>) process...</td></tr>
<tr><td colspan="2">(OR)</td><td rowspan="2">1. Stop <code>mysqld</code>

2. Replace <code>mysqld</code> binary with new version

3. Start <code>mysqld</code></td><td rowspan="2">1. Stop <code>mysqld</code>

2. Make desired changes in hardware, operating system, or both

3. Start <code>mysqld</code></td><td colspan="2">(OR)</td></tr>
<tr><td>1. Stop <code>mysqld</code>

2. Start <code>mysqld</code></td><td>Restart <code>mysqld</code></td><td>1. Stop <code>mysqld</code>

2. Start <code>mysqld</code></td><td>Restart <code>mysqld</code></td></tr>
</table>

In the previous diagram, the **Stop** and **Start** steps indicate that the process must be stopped completely using a shell command (such as `kill` on most Unix systems) or the management client `STOP` command, then started again from a system shell by invoking the `ndbd` or `ndb_mgmd` executable as appropriate. **Restart** indicates the process may be restarted using the `ndb_mgm` management client `RESTART` command.

Important

When performing an upgrade or downgrade of the cluster software, you *must* upgrade or downgrade the management nodes *first*, then the data nodes, and finally the SQL nodes. Doing so in any other order may leave the cluster in an unusable state.

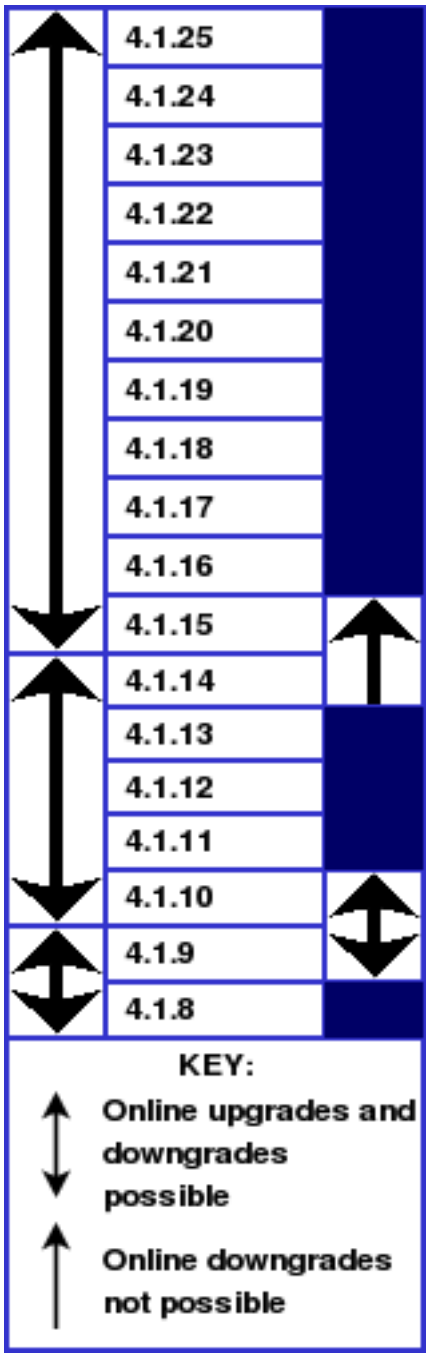
15.2.6.2 MySQL Cluster 4.1 Upgrade and Downgrade Compatibility

This section provides information about MySQL Cluster software and table file compatibility between MySQL 4.1 releases with regard to performing upgrades and downgrades.

Important

Only compatibility between MySQL versions with regard to `NDBCLUSTER` is taken into account in this section, and there are likely other issues to be considered. *As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the MySQL Cluster software*. See Section 2.11.1, “Upgrading MySQL”.

The following table shows Cluster upgrade and downgrade compatibility between different releases of MySQL 4.1:



Notes.

- You cannot perform an online upgrade directly from 4.1.8 to 4.1.10 (or newer); you must first upgrade from 4.1.8 to 4.1.9, then upgrade to 4.1.10. Similarly, you cannot downgrade directly from 4.1.10 (or newer) to 4.1.8; you must first downgrade from 4.1.10 to 4.1.9, then downgrade from 4.1.9 to 4.1.8.
- Online upgrades from MySQL Cluster versions previous to 4.1.8 are not supported; when upgrading from these, you must dump all `NDBCLUSTER` tables using `mysqldump`, install the new version of the software, and then reload the tables from the dump.

- If you wish to upgrade a MySQL Cluster to 4.1.15, you must upgrade to 4.1.14 first, and you must upgrade to 4.1.15 before upgrading to 4.1.16 or newer.
- Cluster downgrades from 4.1.15 to 4.1.14 (or earlier versions) are not supported.
- Direct upgrades or downgrades between MySQL Cluster 4.1 and 5.0 are not supported; you must dump all `NDBCLUSTER` tables using `mysqldump`, install the new version of the software, and then reload the tables from the dump.

15.3 MySQL Cluster Configuration

A MySQL server that is part of a MySQL Cluster differs in one chief respect from a normal (nonclustered) MySQL server, in that it employs the `NDBCLUSTER` storage engine. This engine is also referred to simply as `NDB`, and the two forms of the name are synonymous.

To avoid unnecessary allocation of resources, the server is configured by default with the `NDB` storage engine disabled. To enable `NDB`, you must modify the server's `my.cnf` configuration file, or start the server with the `--ndbcluster` [1301] option.

For more information about `--ndbcluster` [1301] and other MySQL server options specific to MySQL Cluster, see Section 15.3.4.2, “`mysqld` Command Options for MySQL Cluster”.

The MySQL server is a part of the cluster, so it also must know how to access an MGM node to obtain the cluster configuration data. The default behavior is to look for the MGM node on `localhost`. However, should you need to specify that its location is elsewhere, this can be done in `my.cnf` or on the MySQL server command line. Before the `NDB` storage engine can be used, at least one MGM node must be operational, as well as any desired data nodes.

`NDB`, the MySQL Cluster storage engine, is available in binary distributions for Linux, Mac OS X, and Solaris. We are working to support MySQL Cluster on all operating systems supported by MySQL, including Windows. For information about installing MySQL Cluster, see Section 15.2.1, “MySQL Cluster Multi-Computer Installation”.

15.3.1 Quick Test Setup of MySQL Cluster

To familiarize you with the basics, we will describe the simplest possible configuration for a functional MySQL Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as `/var/lib/mysql-cluster`, by executing the following command as the system `root` user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named `config.ini` that contains the following information. Substitute appropriate values for `HostName` and `DataDir` as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: [db], [api], and [mgm] are aliases for [ndbd], [mysqld], and [ndb_mgmd],
```

```
# respectively. [db] is deprecated and should not be used in new installations.

[ndbd default]
NoOfReplicas= 1

[mysqld  default]
[ndb_mgmd default]
[tcp default]

[ndb_mgmd]
HostName= myhost.example.com

[ndbd]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[mysqld]
[mysqld]
[mysqld]
```

You can now start the `ndb_mgmd` management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke `ndb_mgmd`:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single data node by running `ndbd`:

```
shell> ndbd
```

For command-line options which can be used when starting `ndbd`, see Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

By default, `ndbd` looks for the management server at `localhost` on port 1186. (Prior to MySQL 4.1.8, the default port was 2200.)

Note

If you have installed MySQL from a binary tarball, you will need to specify the path of the `ndb_mgmd` and `ndbd` servers explicitly. (Normally, these will be found in `/usr/local/mysql/bin`.)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the `NDBCLUSTER` storage engine is enabled:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.25-Max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
*************************** 12. row ***************************
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an `NDBCLUSTER` table:

```
shell> mysql
mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
*************************** 1. row ***************************
       Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client:

```
shell> ndb_mgm
```

Use the `SHOW` command from within the management client to obtain a report on the cluster's status:

```
ndb_mgm> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     1 node(s)
id=2    @127.0.0.1  (Version: 3.5.3, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @127.0.0.1  (Version: 3.5.3)

[mysqld(API)]   3 node(s)
id=3    @127.0.0.1  (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

At this point, you have successfully set up a working MySQL Cluster. You can now store data in the cluster by using any table created with `ENGINE=NDBCLUSTER` or its alias `ENGINE=NDB`.

15.3.2 MySQL Cluster Configuration Files

Configuring MySQL Cluster requires working with two files:

- `my.cnf`: Specifies options for all MySQL Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- `config.ini`: This file, sometimes known as the *global configuration file*, is read only by the MySQL Cluster management server, which then distributes the information contained therein to all processes participating in the cluster. `config.ini` contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see Sections of the `config.ini` File [1242].

We are continuously making improvements in Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we will try to let Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in Section 1.8, “How to Report Bugs or Problems”.

15.3.2.1 MySQL Cluster Configuration: Basic Example

To support MySQL Cluster, you will need to update `my.cnf` as shown in the following example. You may also specify these parameters on the command line when invoking the executables.

In MySQL 4.1.8, some simplifications in `my.cnf` were introduced, including new sections for `NDBCLUSTER` executables.

Note

The options shown here should not be confused with those that are used in `config.ini` global configuration files. Global configuration options are discussed later in this section.

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid from 4.1.8)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com


# provide connectstring for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(For more information on connectstrings, see Section 15.3.2.2, “The MySQL Cluster Connectstring”.)

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Important

Once you have started a `mysqld` process with the `NDBCLUSTER` and `ndb-connectstring` parameters in the `[mysqld]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design*.

Starting with MySQL 4.1.8, you may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

For additional `NDB` variables that can be set in the `my.cnf` file, see Section 15.3.4.3, “MySQL Cluster System Variables”.

The MySQL Cluster global configuration file is named `config.ini` by default. It is read by `ndb_mgmd` at startup and can be placed anywhere. Its location and name are specified by using `--config-file=path_name` on the `ndb_mgmd` command line. If the configuration file is not specified, `ndb_mgmd` by default tries to read a file named `config.ini` located in the current working directory.

The global configuration file for MySQL Cluster uses INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (“`:`”) as well as the equal sign (“`=`”); however, the equal sign is preferred. Another deviation is that sections are not uniquely identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. (*Exception*: The `NoOfReplicas` configuration parameter has no default value, and must always be specified explicitly in the `[ndbd default]` section.) To create a default value section, simply add the word `default` to the section name. For example, an `[ndbd]` section contains parameters that apply to a particular data node, whereas an `[ndbd default]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory size. To configure them all, create an `[ndbd default]` section that contains a `DataMemory` line to specify the data memory size.

The global configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two MySQL servers is shown here:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
```

```
# can be started only on the host mysqld_5.mysql.com

[ndbd default]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[ndb_mgmd]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[ndbd]
HostName= ndbd_2.mysql.com

[ndbd]
HostName= ndbd_3.mysql.com

[mysqld]
[mysqld]
HostName= mysqld_5.mysql.com
```

Each node has its own section in the `config.ini` file. For example, this cluster has two data nodes, so the preceding configuration file contains two `[ndbd]` sections defining these nodes.

Note

Do not place comments on the same line as a section heading in the `config.ini` file; this causes the management server not to start because it cannot parse the configuration file in such cases.

Sections of the `config.ini` File

There are six different sections that you can use in the `config.ini` configuration file, as described in the following list:

- `[computer]`: Defines cluster hosts. This is not required to configure a viable MySQL Cluster, but be may used as a convenience when setting up a large cluster. See Section 15.3.2.3, “Defining Computers in a MySQL Cluster”, for more information.
- `[ndbd]`: Defines a cluster data node (`ndbd` process). See Section 15.3.2.5, “Defining MySQL Cluster Data Nodes”, for details.
- `[mysqld]`: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see Section 15.3.2.6, “Defining SQL and Other API Nodes in a MySQL Cluster”.
- `[mgm]` or `[ndb_mgmd]`: Defines a cluster management server (MGM) node. For information concerning the configuration of MGM nodes, see Section 15.3.2.4, “Defining a MySQL Cluster Management Server”.
- `[tcp]`: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default connection protocol. Normally, `[tcp]` or `[tcp default]` sections are not required to set up a MySQL Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See Section 15.3.2.7, “MySQL Cluster TCP/IP Connections”, for information about available TCP/IP configuration parameters and how to use them. (You may also find Section 15.3.2.8, “MySQL Cluster TCP/IP Connections Using Direct Connections” to be of interest in some cases.)
- `[shm]`: Defines shared-memory connections between nodes. Prior to MySQL 4.1.9, this type of connection was available only in binaries that were built using the `--with-ndb-shm` option. Beginning with MySQL 4.1.9-max, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see Section 15.3.2.9, “MySQL Cluster Shared-Memory Connections”.

- `[sci]`:Defines *Scalable Coherent Interface* connections between cluster data nodes. Such connections require software which, while freely available, is not part of the MySQL Cluster distribution, as well as specialized hardware. See Section 15.3.2.10, “SCI Transport Connections in MySQL Cluster” for detailed information about SCI interconnects.

You can define `default` values for each section. As of MySQL 4.1.5, all parameter names are case-insensitive, which differs from parameters specified in `my.cnf` or `my.ini` files.

15.3.2.2 The MySQL Cluster Connectstring

With the exception of the MySQL Cluster management server (`ndb_mgmd`), each node that is part of a MySQL Cluster requires a *connectstring* that points to the management server's location. This connectstring is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connectstring is as follows:

```
[nodeid=node_id, ]host-definition[, host-definition[, ...]]

host-definition:
    host_name[:port_number]
```

`node_id` is an integer larger than 1 which identifies a node in `config.ini`. *`host_name`* is a string representing a valid Internet host name or IP address. *`port_number`* is an integer referring to a TCP/IP port number.

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short):   "myhost1"
```

`localhost:1186` is used as the default connectstring value if none is provided. If *`port_num`* is omitted from the connectstring, the default port is 1186.

Note

Prior to MySQL 4.1.8, the default port was 2200.

Port 1186 should always be available on the network because it has been assigned by IANA for this purpose (see http://www.iana.org/assignments/port-numbers for details).

By listing multiple host definitions, it is possible to designate several redundant management servers. A MySQL Cluster data or API node attempts to contact successive management servers on each host in the order specified, until a successful connection has been established.

There are a number of different ways to specify the connectstring:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)
- Beginning with MySQL 4.1.8, it is also possible to set the connectstring for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.
- For backward compatibility, two other options are available, using the same syntax:

  1. Set the `NDB_CONNECTSTRING` environment variable to contain the connectstring.
  2. Write the connectstring for each executable into a text file named `Ndb.cfg` and place this file in the executable's startup directory.

  However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connectstring is to set it on the command line or in the `my.cnf` file for each executable.

The maximum length of a connectstring is 1024 characters.

15.3.2.3 Defining Computers in a MySQL Cluster

The `[computer]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- `Id`

Introduced	4.1.3	
Restart Type	initial, system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

This is a unique identifier, used to refer to the host computer elsewhere in the configuration file.

Important

The computer ID is *not* the same as the node ID used for a management, API, or data node. Unlike the case with node IDs, you cannot use `NodeId` in place of `Id` in the `[computer]` section of the `config.ini` file.

- `HostName`

Introduced	4.1.3	
Restart Type	system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

This is the computer's hostname or IP address.

15.3.2.4 Defining a MySQL Cluster Management Server

The `[ndb_mgmd]` section is used to configure the behavior of the management server. `[mgm]` can be used as an alias; the two section names are equivalent. All parameters in the following list are optional and assume their default values if omitted.

Note

If neither the `ExecuteOnComputer` nor the `HostName` parameter is present, the default value `localhost` will be assumed for both.

- `Id`

Introduced	4.1.3	
Restart Type	initial, system	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`1 .. 63`

Each node in the cluster has a unique identity, which is represented by an integer value in the range 1 to 63 inclusive. This ID is used by all internal cluster messages for addressing the node.

This parameter can also be written as `NodeId`, although the short form is sufficient (and preferred for this reason).

- `ExecuteOnComputer`

Introduced	4.1.3	
Restart Type	system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

This refers to the `Id` set for one of the computers defined in a `[computer]` section of the `config.ini` file.

- `PortNumber`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3, <= 4.1.7)	
	Type	`numeric`
	Default	`2200`
	Range	`0 .. 64K`
	Permitted Values (>= 4.1.8)	
	Type	`numeric`
	Default	`1186`
	Range	`0 .. 64K`

This is the port number on which the management server listens for configuration requests and management commands.

- `HostName`

Introduced	4.1.3

Restart Type	system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `LogDestination`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	`FILE: filename=ndb_nodeid_cluster.log, maxsize=1000000, maxfiles=6`
	Range	`..`

This parameter specifies where to send cluster logging information. There are three options in this regard—`CONSOLE`, `SYSLOG`, and `FILE`—with `FILE` being the default:

- `CONSOLE` outputs the log to `stdout`:

```
CONSOLE
```

- `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.

Note

Not every facility is necessarily supported by every operating system.

```
SYSLOG:facility=syslog
```

- `FILE` pipes the cluster log output to a regular file on the same machine. The following values can be specified:
  - `filename`: The name of the log file.
  - `maxsize`: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old log file is renamed by appending `.N` to the file name, where `N` is the next number not yet used with this name.
  - `maxfiles`: The maximum number of log files.

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

The default value for the FILE parameter is FILE:filename=ndb_*node_id*_cluster.log,maxsize=1000000,maxfiles=6, where *node_id* is the ID of the node.

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

- ArbitrationRank

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	1
	Range	0 .. 2

This parameter is used to define which nodes can act as arbitrators. Only management nodes and SQL nodes can be arbitrators. ArbitrationRank can take one of the following values:

- 0: The node will never be used as an arbitrator.
- 1: The node has high priority; that is, it will be preferred as an arbitrator over low-priority nodes.
- 2: Indicates a low-priority node which be used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its ArbitrationRank to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

- ArbitrationDelay

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	0
	Range	0 .. 4G

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- DataDir

Introduced	4.1.3
Restart Type	node

Permitted Values (>= 4.1.3)	
Type	`string`
Default	`.`
Range	`..`

This specifies the directory where output files from the management server will be placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the `FILE` parameter for `LogDestination` as discussed previously in this section.)

The default value for this parameter is the directory in which `ndb_mgmd` is located.

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect.

To add new management servers to a running MySQL Cluster, it is also necessary to perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see Section 15.1.4.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

15.3.2.5 Defining MySQL Cluster Data Nodes

The `[ndbd]` and `[ndbd default]` sections are used to configure the behavior of the cluster's data nodes.

There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameters are:

- Either `ExecuteOnComputer` or `HostName`, which must be defined in the local `[ndbd]` section.
- The parameter `NoOfReplicas`, which must be defined in the `[ndbd default]` section, as it is common to all Cluster data nodes.

Most data node parameters are set in the `[ndbd default]` section. Only those parameters explicitly stated as being able to set local values are permitted to be changed in the `[ndbd]` section. Where present, `HostName`, `Id` and `ExecuteOnComputer` *must* be defined in the local `[ndbd]` section, and not in any other section of `config.ini`. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use `K`, `M`, or `G` as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024. (For example, `100K` means 100 × 1024 = 102400.) Parameter names and values are currently case-sensitive.

Identifying data nodes. The `Id` value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- `Id`

Introduced	4.1.3
Restart Type	initial, system

	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`1 .. 48`

This is the node ID used as the address of the node for all cluster internal messages. For data nodes, this is an integer in the range 1 to 49 inclusive. Each node in the cluster must have a unique identity.

This parameter can also be written as `NodeId`, although the short form is sufficient (and preferred for this reason).

- `ExecuteOnComputer`

Introduced	4.1.3	
Restart Type	system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

This refers to the `Id` set for one of the computers defined in a `[computer]` section.

- `HostName`

Introduced	4.1.3	
Restart Type	system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	`localhost`
	Range	`..`

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `ServerPort` (*OBSOLETE*)

Each node in the cluster uses a port to connect to other nodes. This port is used also for non-TCP transporters in the connection setup phase. The default port is allocated dynamically in such a way as to ensure that no two nodes on the same computer receive the same port number, so it should not normally be necessary to specify a value for this parameter.

- `NoOfReplicas`

Introduced	4.1.3
Restart Type	initial, system
	Permitted Values (>= 4.1.3)

	Type	`numeric`
	Default	
	Range	`1 .. 4`
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`1 .. 4`
	Permitted Values	
	Type	`numeric`
	Default	`2`
	Range	`1 .. 4`
	Permitted Values	
	Type	`numeric`
	Default	`2`
	Range	`1 .. 4`

This global parameter can be set only in the `[ndbd default]` section, and defines the number of replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that `NoOfReplicas` is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to fail.

If no node IDs are provided, the order of the data nodes will be the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client's `SHOW` command.

There is no default value for `NoOfReplicas`; the recommended value is 2 for most common usage scenarios.

The maximum possible value is 4; *currently, only the values 1 and 2 are actually supported* (see Bug #18621).

Important

Setting `NoOfReplicas` to 1 means that there is only a single copy of all Cluster data; in this case, the loss of a single data node causes the cluster to fail because there are no additional copies of the data stored by that node.

The value for this parameter must divide evenly into the number of data nodes in the cluster. For example, if there are two data nodes, then `NoOfReplicas` must be equal to either 1 or 2, since 2/3 and 2/4 both yield fractional values; if there are four data nodes, then `NoOfReplicas` must be equal to 1, 2, or 4.

-

`DataDir`

Introduced	4.1.3	
Restart Type	initial, node	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	`.`
	Range	`..`

This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

The default is the data node process working directory.

- `FileSystemPath`

Restart Type	initial, node	
	Permitted Values	
	Type	`string`
	Default	`DataDir`
	Range	`..`

This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs, and data files are placed. The default is the directory specified by `DataDir`.

Note

This directory must exist before the `ndbd` process is initiated.

The recommended directory hierarchy for MySQL Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's file system is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named `ndb_2_fs`.

- `BackupDataDir`

Introduced	4.1.3	
Restart Type	initial, node	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	`FileSystemPath`
	Range	`..`

This parameter specifies the directory in which backups are placed. If omitted, the default backup location is the directory named `BACKUP` under the location specified by the `FileSystemPath` parameter. (See above.)

`DataMemory` and `IndexMemory` are `[ndbd]` parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how `DataMemory` and `IndexMemory` are used, as they usually need to be updated to reflect actual usage by the cluster:

- `DataMemory`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`80M`
	Range	`1M .. 1024G`

This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

The memory allocated by `DataMemory` is used to store both the actual records and indexes. Each record is currently of fixed size. (Even `VARCHAR` columns are stored as fixed-width columns.) There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page.

The maximum record size is currently 8052 bytes.

The memory space defined by `DataMemory` is also used to store ordered indexes, which use about 10 bytes per record. Each table row is represented in the ordered index. A common error among users is to assume that all indexes are stored in the memory allocated by `IndexMemory`, but this is not the case: Only primary key and unique hash indexes use this memory; ordered indexes use the memory allocated by `DataMemory`. However, creating a primary key or unique hash index also creates an ordered index on the same keys, unless you specify `USING HASH` in the index creation statement. This can be verified by running `ndb_desc -d db_name table_name` in the management client.

The memory space allocated by `DataMemory` consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in `NoOfReplicas`.

In addition, due to the way in which new pages are allocated when the capacity of the current page is exhausted, there is an additional overhead of approximately 18.75%. When more `DataMemory` is required, more than one new page is allocated, according to the following formula:

```
number of new pages = FLOOR(number of current pages × 0.1875) + 1
```

For example, if 15 pages are currently allocated to a given table and an insert to this table requires additional storage space, the number of new pages allocated to the table is `FLOOR(15 × 0.1875) + 1 = FLOOR(2.8125) + 1 = 2 + 1 = 3`. Now 15 + 3 = 18 memory pages are allocated to the table. When the last of these 18 pages becomes full, `FLOOR(18 × 0.1875) + 1 = FLOOR(3.3750) + 1 = 3 + 1 = 4` new pages are allocated, so the total number of pages allocated to the table is now 22.

Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. (This also means that `DataMemory` pages, once allocated to a given table, cannot be used by other tables.) Performing a node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The `DataMemory` memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the `DataMemory`. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for `DataMemory` is 80MB; the minimum is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2–4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason.

- `IndexMemory`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`18M`
	Range	`1M .. 1T`

This parameter controls the amount of storage used for hash indexes in MySQL Cluster. Hash indexes are always used for primary key indexes, unique indexes, and unique constraints. Note that when defining a primary key and a unique index, two indexes will be created, one of which is a hash index used for all tuple accesses as well as lock handling. It is also used to enforce unique constraints.

The size of the hash index is 25 bytes per record, plus the size of the primary key. For primary keys larger than 32 bytes another 8 bytes is added.

The default value for `IndexMemory` is 18MB. The minimum is 1MB.

- `StringMemory`

Introduced	4.1.3
Restart Type	system

Permitted Values (>= 4.1.3)	
Type	numeric
Default	0
Range	0 .. 4G

This parameter determines how much memory is allocated for strings such as table names, and is specified in an [ndbd] or [ndbd default] section of the config.ini file. A value between 0 and 100 inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum size of .FRM files, MaxNoOfTriggers, maximum column name size, and maximum default column value. In general it is safe to assume that the maximum default value is approximately 5 MB for a MySQL Cluster having 1000 tables.

A value greater than 100 is interpreted as a number of bytes.

In MySQL 4.1, the default value is 100—that is, 100 percent of the default maximum, or roughly 5 MB. It is possible to reduce this value safely, but it should never be less than 5 percent. If you encounter Error 773 Out of string memory, please modify StringMemory config parameter: Permanent error: Schema error, this means that means that you have set the StringMemory value too low. 25 (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions, as when there are hundreds or thousands of NDB tables with names whose lengths and columns whose number approach their permitted maximums.

The following example illustrates how memory is used for a table. Consider this table definition:

```
CREATE TABLE example (
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. Having no nullable columns saves 4 bytes of overhead. In addition, we have two ordered indexes on columns a and b consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with b as primary key and a as a column. This other table consumes an additional 29 bytes of index memory per record in the example table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in MySQL Cluster to handle uniqueness constraints.

Currently, the only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both IndexMemory and DataMemory is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two replicas, there will be two node groups. Thus, the total data memory available is 2 × DataMemory for each data node.

It is highly recommended that `DataMemory` and `IndexMemory` be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

`DataMemory` and `IndexMemory` can be changed, but decreasing either of these can be risky; doing so can easily lead to a node or even an entire MySQL Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by restarting each data node in turn.

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

Transaction parameters. The next three `[ndbd]` parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` sets the number of parallel transactions possible in a node. `MaxNoOfConcurrentOperations` sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations`) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

- `MaxNoOfConcurrentTransactions`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`4096`
	Range	`32 .. 4G`

Each cluster data node requires a transaction record for each active transaction in the cluster. The task of coordinating transactions is distributed among all of the data nodes. The total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual MySQL servers. Each connection to a MySQL server requires at least one transaction record, plus an additional transaction object per table accessed by that connection. This means that a reasonable minimum for this parameter is

```
MaxNoOfConcurrentTransactions =
    (maximum number of tables accessed in any single transaction + 1)
    * number of cluster SQL nodes
```

Suppose that there are 4 SQL nodes using the cluster. A single join involving 5 tables requires 6 transaction records; if there are 5 such joins in a transaction, then 5 * 6 = 30 transaction records are required for this transaction, per MySQL server, or 30 * 4 = 120 transaction records total.

This parameter must be set to the same value for all cluster data nodes. This is due to the fact that, when a data node fails, the oldest surviving node re-creates the transaction state of all transactions that were ongoing in the failed node.

Changing the value of `MaxNoOfConcurrentTransactions` requires a complete shutdown and restart of the cluster.

The default value is 4096.

- `MaxNoOfConcurrentOperations`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`32K`
	Range	`32 .. 4G`

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions of only a few operations each and not involving a great many records, there is no need to set this parameter very high. When performing large transactions involving many records need to set this parameter higher.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle 1,000,000 concurrent updates using transactions, you should set this value to 1000000 / 4 = 250000.

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. `MaxNoOfConcurrentOperations` will always be used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

-

`MaxNoOfLocalOperations`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`UNDEFINED`
	Range	`32 .. 4G`

By default, this parameter is calculated as 1.1 × `MaxNoOfConcurrentOperations`. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

Transaction temporary storage. The next set of `[ndbd]` parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

- `MaxNoOfConcurrentIndexOperations`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`8K`
	Range	`0 .. 4G`

For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter `MaxNoOfConcurrentOperations`.

The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

- `MaxNoOfFiredTriggers`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	

	Type	numeric
	Default	4000
	Range	0 .. 4G

The default value of MaxNoOfFiredTriggers is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- TransactionBufferMemory

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	1M
	Range	1K .. 4G

The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for TransactionBufferMemory is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter ZATTRBUF_FILESIZE (found in ndb/src/kernel/blocks/Dbtc/Dbtc.hpp) set to 4000 × 128 bytes (500KB). A similar buffer for key information, ZDATABUF_FILESIZE (also in Dbtc.hpp) contains 4000 × 16 = 62.5KB of buffer space. Dbtc is the module that handles transaction coordination.

Scans and buffering. There are additional [ndbd] parameters in the Dblqh module (in ndb/src/kernel/blocks/Dblqh/Dblqh.hpp) that affect reads and updates. These include ZATTRINBUF_FILESIZE, set by default to 10000 × 128 bytes (1250KB) and ZDATABUF_FILE_SIZE, set by default to 10000*16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- MaxNoOfConcurrentScans

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	256

	Range	`2 .. 500`

This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain `MaxNoOfConcurrentScans` scans concurrently from all nodes in the cluster.

Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

The default value of `MaxNoOfConcurrentScans` is 256. The maximum value is 500.

- `MaxNoOfLocalScans`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`MaxNoOfConcurrentScans * [# of data nodes] + 2`
	Range	`32 .. 4G`

Specifies the number of local scan records if many scans are not fully parallelized. If the number of local scan records is not provided, it is calculated as the product of `MaxNoOfConcurrentScans` and the number of data nodes in the system. The minimum value is 32.

- `BatchSizePerLocalScan`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`64`
	Range	`1 .. 992`

This parameter is used to calculate the number of lock records used to handle concurrent scan operations.

The default value is 64; this value has a strong connection to the `ScanBatchSize` defined in the SQL nodes.

- `LongMessageBuffer`

Introduced	4.1.3	

Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`1M`
	Range	`512K .. 4G`
	Permitted Values	
	Type	`numeric`
	Default	`4M`
	Range	`512K .. 4G`

This is an internal buffer used for passing messages within individual nodes and between nodes. Although it is highly unlikely that this would need to be changed, it is configurable. By default, it is set to 1MB.

Logging and checkpointing. The following `[ndbd]` parameters control log and checkpoint behavior.

- `NoOfFragmentLogFiles`

Introduced	4.1.3	
Restart Type	initial, node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`8`
	Range	`3 .. 4G`

This parameter sets the number of REDO log files for the node, and thus the amount of space allocated to REDO logging. Because the REDO log files are organized in a ring, it is extremely important that the first and last log files in the set (sometimes referred to as the “head” and “tail” log files, respectively) do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A `REDO` log record is not removed until three local checkpoints have been completed since that log record was inserted. Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

How these parameters interact and proposals for how to configure them are discussed in Section 15.3.2.11, “Configuring MySQL Cluster Parameters for Local Checkpoints”.

The default parameter value is 8, which means 8 sets of 4 16MB files for a total of 512MB. In other words, REDO log space is always allocated in blocks of 64MB. In scenarios requiring a great many updates, the value for `NoOfFragmentLogFiles` may need to be set as high as 300 or even higher to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (`Out of log file space temporarily`). This condition prevails until a checkpoint has completed and the log tail can be moved forward.

Important

This parameter cannot be changed “on the fly”; you must restart the node using `--initial`. If you wish to change this value for all data nodes in a running cluster, you can do so using a rolling node restart (using `--initial` when starting each data node).

- `MaxNoOfOpenFiles`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`40`
	Range	`20 .. 4G`

This parameter sets a ceiling on how many internal threads to allocate for open files. *Any situation requiring a change in this parameter should be reported as a bug*.

The default value is 40.

- `MaxNoOfSavedMessages`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`25`
	Range	`0 .. 4G`

This parameter sets the maximum number of trace files that are kept before overwriting old ones. Trace files are generated when, for whatever reason, the node crashes.

The default is 25 trace files.

Metadata objects. The next set of `[ndbd]` parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters. Note that these act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

- `MaxNoOfAttributes`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`

	Default	`1000`
	Range	`32 .. 4G`

Defines the number of attributes that can be defined in the cluster.

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting `MaxNoOfAttributes`, it is important to prepare in advance for any `ALTER TABLE` statements that you might want to perform in the future. This is due to the fact, during the execution of `ALTER TABLE` on a Cluster table, 3 times the number of attributes as in the original table are used, and a good practice is to permit double this amount. For example, if the MySQL Cluster table having the greatest number of attributes (*`greatest_number_of_attributes`*) has 100 attributes, a good starting point for the value of `MaxNoOfAttributes` would be `6 * greatest_number_of_attributes = 600`.

You should also estimate the average number of attributes per table and multiply this by the total number of MySQL Cluster tables. If this value is larger than the value obtained in the previous paragraph, you should use the larger value instead.

Assuming that you can create all desired tables without any problems, you should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` by another multiple of `MaxNoOfTables` and test it again.

- `MaxNoOfTables`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`128`
	Range	`8 .. 1600`

A table object is allocated for each table and for each unique hash index in the cluster. This parameter sets the maximum number of table objects for the cluster as a whole.

For each attribute that has a `BLOB` data type an extra table is used to store most of the `BLOB` data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 1600. Each table object consumes approximately 20KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfOrderedIndexes`

Introduced	4.1.3	

Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`128`
	Range	`0 .. 4G`

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index. `MaxNoOfOrderedIndexes` sets the total number of hash indexes that can be in use in the system at any one time.

The default value of this parameter is 128. Each hash index object consumes approximately 10KB of data per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfUniqueHashIndexes`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`64`
	Range	`0 .. 4G`

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the `USING HASH` option when defining the unique index.

The default value is 64. Each index consumes approximately 15KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfTriggers`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`768`
	Range	`0 .. 4G`

Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

This parameter sets the maximum number of trigger objects in the cluster.

The default value is 768.

- `MaxNoOfIndexes`

  This parameter was deprecated in MySQL 4.1.5. In MySQL 4.1.6 and newer versions; you should use `MaxNoOfOrderedIndexes` and `MaxNoOfUniqueHashIndexes` instead.

  This parameter is used only by unique hash indexes. There needs to be one record in this pool for each unique hash index defined in the cluster.

  The default value of this parameter is 128.

Boolean parameters. The behavior of data nodes is also affected by a set of `[ndbd]` parameters taking on boolean values. These parameters can each be specified as `TRUE` by setting them equal to `1` or `Y`, and as `FALSE` by setting them equal to `0` or `N`.

- `LockPagesInMainMemory`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`boolean`
	Default	`0`
	Range	`0 .. 1`

  For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

  This feature is disabled by default.

  Note

  To make use of this parameter, the data node process must be run as system root.

- `StopOnError`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`boolean`
	Default	`true`

	Valid Values	`true`
		`false`

This parameter specifies whether an `ndbd` process should exit or perform an automatic restart when an error condition is encountered.

This feature is enabled by default.

- `Diskless`

Introduced	4.1.3	
Restart Type	initial, system	
	Permitted Values (>= 4.1.3)	
	Type	`boolean`
	Default	`false`
	Valid Values	`true`
		`false`

It is possible to specify MySQL Cluster tables as *diskless*, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run `ndbd` on a diskless computer.

Important

This feature causes the *entire* cluster to operate in diskless mode.

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

`Diskless` is disabled by default.

- `RestartOnErrorInsert`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`2`
	Range	`0 .. 4`

This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

This feature is disabled by default.

Controlling Timeouts, Intervals, and Disk Paging

There are a number of [ndbd] parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- TimeBetweenWatchDogCheck

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	6000
	Range	70 .. 4G

To prevent the main thread from getting stuck in an endless loop at some point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

The default timeout is 4000 milliseconds (4 seconds).

- StartPartialTimeout

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	30000
	Range	0 .. 4G

This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default value is 30000 milliseconds (30 seconds). 0 disables the timeout, in which case the cluster may start only if all nodes are available.

- StartPartitionedTimeout

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	60000
	Range	0 .. 4G

If the cluster is ready to start after waiting for `StartPartialTimeout` milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed. If `StartPartitionedTimeout` is set to 0, the cluster waits indefinitely.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default timeout is 60000 milliseconds (60 seconds).

- `StartFailureTimeout`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 4G`

If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 (the default value) means that no data node timeout is applied.

For nonzero values, this parameter is measured in milliseconds. For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node containing several gigabytes of data, a period as long as 10–15 minutes (that is, 600000 to 1000000 milliseconds) might be required to perform a node restart.

- `HeartbeatIntervalDbDb`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`1500`
	Range	`10 .. 4G`

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. After missing three heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is four times the heartbeat interval.

The default heartbeat interval is 1500 milliseconds (1.5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node will be declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

- `HeartbeatIntervalDbApi`

Introduced	4.1.3

Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	1500
	Range	100 .. 4G

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for HeartbeatIntervalDbDb.

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

- TimeBetweenLocalCheckpoints

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	20
	Range	0 .. 31

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the default value 20 means 4MB (4×2^{20}) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting TimeBetweenLocalCheckpoints to 6 or less means that local checkpoints will be executed continuously without pause, independent of the cluster's workload.

- TimeBetweenGlobalCheckpoints

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	2000
	Range	10 .. 32000

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases—a complete crash of the cluster—is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- `TimeBetweenInactiveTransactionAbortCheck`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`1000`
	Range	`1000 .. 4G`

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction will be checked for timing out once per second.

The default value is 1000 milliseconds (1 second).

- `TransactionInactiveTimeout`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`4G`
	Range	`0 .. 4G`

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is zero (no timeout). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a relatively small value. The unit is milliseconds.

- `TransactionDeadlockDetectionTimeout`

Introduced	4.1.3

Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`1200`
	Range	`50 .. 4G`

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. A failure to respond can occur for any of the following reasons:

- The node is “dead”
- The operation has entered a lock queue
- The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection. In MySQL 4.1.18 and earlier versions, setting it too high could cause undesirable behavior in situations involving deadlocks and node failure. Beginning with MySQL 4.1.19, active transactions occurring during node failures are actively aborted by the Cluster Transaction Coordinator, and so high settings are no longer an issue with this parameter.

The default timeout value is 1200 milliseconds (1.2 seconds). The effective minimum value is 100 milliseconds; it is possible to set it as low as 50 milliseconds, but any such value is treated as 100 ms. (Bug #44099)

- `NoOfDiskPagesToDiskAfterRestartTUP`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`40`
	Range	`1 .. 4G`

When executing a local checkpoint, the algorithm flushes all data pages to disk. Merely doing so as quickly as possible without any moderation is likely to impose excessive loads on processors, networks, and disks. To control the write speed, this parameter specifies how many pages per 100 milliseconds are to be written. In this context, a “page” is defined as 8KB. This parameter is specified in units of 80KB per second, so setting `NoOfDiskPagesToDiskAfterRestartTUP` to a value of `20` entails writing 1.6MB in data pages to disk each second during a local checkpoint. This value includes the writing of UNDO log records for data pages. That is, this parameter handles the limitation of writes from data memory. UNDO log records for index pages are handled by the parameter `NoOfDiskPagesToDiskAfterRestartACC`. (See the entry for `IndexMemory` for information about index pages.)

In short, this parameter specifies how quickly to execute local checkpoints. It operates in conjunction with `NoOfFragmentLogFiles`, `DataMemory`, and `IndexMemory`.

For more information about the interaction between these parameters and possible strategies for choosing appropriate values for them, see Section 15.3.2.11, “Configuring MySQL Cluster Parameters for Local Checkpoints”.

The default value is 40 (3.2MB of data pages per second).

- `NoOfDiskPagesToDiskAfterRestartACC`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`20`
	Range	`1 .. 4G`

This parameter uses the same units as `NoOfDiskPagesToDiskAfterRestartTUP` and acts in a similar fashion, but limits the speed of writing index pages from index memory.

The default value of this parameter is 20 (1.6MB of index memory pages per second).

- `NoOfDiskPagesToDiskDuringRestartTUP`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`40`
	Range	`1 .. 4G`

This parameter is used in a fashion similar to `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, only it does so with regard to local checkpoints executed in the node when a node is restarting. A local checkpoint is always performed as part of all node restarts. During a node restart it is possible to write to disk at a higher speed than at other times, because fewer activities are being performed in the node.

This parameter covers pages written from data memory.

The default value is 40 (3.2MB per second).

- `NoOfDiskPagesToDiskDuringRestartACC`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`20`

	Range	`1 .. 4G`

Controls the number of index memory pages that can be written to disk during the local checkpoint phase of a node restart.

As with `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, values for this parameter are expressed in terms of 8KB pages written per 100 milliseconds (80KB/second).

The default value is 20 (1.6MB per second).

- `ArbitrationTimeout`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`3000`
	Range	`10 .. 4G`

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

The default value is 1000 milliseconds (1 second).

Buffering and logging. Several `[ndbd]` configuration parameters corresponding to former compile-time parameters were introduced in MySQL 4.1.5. These enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the `NDB` storage engine's file system abstraction layer.

- `UndoIndexBuffer`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`2M`
	Range	`1M .. 4G`

The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The `NDB` storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 (`Index UNDO buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- `UndoDataBuffer`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`16M`
	Range	`1M .. 4G`

This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 (`Data UNDO buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- `RedoBuffer`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`8M`
	Range	`1M .. 4G`

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

`RedoBuffer` sets the size of the buffer in which the REDO log is written, and is 8MB by default. The minimum value is 1MB.

If this buffer is too small, the NDB storage engine issues error code 1221 (`REDO log buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

Controlling log messages. In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to `stdout`. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to `stdout`; setting it to 0 means that there will be no event reports made in that category.

By default, only the startup message is sent to `stdout`, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- `LogLevelStartup`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`1`
	Range	`0 .. 15`

The reporting level for events generated during startup of the process.

The default level is 1.

-

`LogLevelShutdown`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 15`

The reporting level for events generated as part of graceful shutdown of a node.

The default level is 0.

- `LogLevelStatistic`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 15`

The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

The default level is 0.

- `LogLevelCheckpoint`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 15`

The reporting level for events generated by local and global checkpoints.

The default level is 0.

- `LogLevelNodeRestart`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	

	Type	numeric
	Default	0
	Range	0 .. 15

The reporting level for events generated during node restart.

The default level is 0.

- LogLevelConnection

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	0
	Range	0 .. 15

The reporting level for events generated by connections between cluster nodes.

The default level is 0.

- LogLevelError

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	0
	Range	0 .. 15

The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

The default level is 0.

- LogLevelInfo

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	0
	Range	0 .. 15

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.

Backup parameters. The [ndbd] parameters discussed in this section define memory buffers set aside for execution of online backups.

- BackupDataBufferSize

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	2M
	Range	0 .. 4G

In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as BackupWriteSize (see below), the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup process pauses the scan and waits until some disk writes have completed freed up memory so that scanning may continue.

The default value is 2MB.

- BackupLogBufferSize

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	2M
	Range	0 .. 4G

The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log buffer must be large enough to handle the load caused by write activities while the backup is being made. See Section 15.5.3.3, “Configuration for MySQL Cluster Backups”.

The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

The default value is 2MB.

- `BackupMemory`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`4M`
	Range	`0 .. 4G`

This parameter is simply the sum of `BackupDataBufferSize` and `BackupLogBufferSize`.

The default value is 2MB + 2MB = 4MB.

Important

If `BackupDataBufferSize` and `BackupLogBufferSize` taken together exceed 4MB, then this parameter must be set explicitly in the `config.ini` file to their sum.

- `BackupWriteSize`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`32K`
	Range	`2K .. 4G`

This parameter specifies the default size of messages written to disk by the backup log and backup data buffers.

The default value is 32KB.

- `BackupMaxWriteSize`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`256K`
	Range	`2K .. 4G`

This parameter specifies the maximum size of messages written to disk by the backup log and backup data buffers.

The default value is 256KB.

Important

When specifying these parameters, the following relationships must hold true. Otherwise, the data node will be unable to start.

- `BackupDataBufferSize >= BackupWriteSize + 188KB`
- `BackupLogBufferSize >= BackupWriteSize + 16KB`
- `BackupMaxWriteSize >= BackupWriteSize`

Note

To add new data nodes to a MySQL Cluster, it is necessary to shut down the cluster completely, update the `config.ini` file, and then restart the cluster (that is, you must perform a system restart). All data node processes must be started with the `--initial` option.

We are working to make it possible to add new data node groups to a running cluster online in a future release; however, we do not plan to implement this change in MySQL 4.1.

15.3.2.6 Defining SQL and Other API Nodes in a MySQL Cluster

The `[mysqld]` and `[api]` sections in the `config.ini` file define the behavior of the MySQL servers (SQL nodes) and other applications (API nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL or API node.

Generally speaking, a `[mysqld]` section is used to indicate a MySQL server providing an SQL interface to the cluster, and an `[api]` section is used for applications other than `mysqld` processes accessing cluster data, but the two designations are actually synonomous; you can, for instance, list parameters for a MySQL server acting as an SQL node in an `[api]` section.

Note

For a discussion of MySQL server options for MySQL Cluster, see Section 15.3.4.2, “`mysqld` Command Options for MySQL Cluster”; for information about MySQL server system variables relating to MySQL Cluster, see Section 15.3.4.3, “MySQL Cluster System Variables”.

- `Id`

Introduced	4.1.3	
Restart Type	initial, system	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`1 .. 63`

The `Id` is an integer value used to identify the node in all cluster internal messages. It must be an integer in the range 1 to 63 inclusive, and must be unique among all node IDs within the cluster.

This parameter can also be written as `NodeId`, although the short form is sufficient (and preferred for this reason).

- `ExecuteOnComputer`

Introduced	4.1.3	
Restart Type	system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

This refers to the `Id` set for one of the computers (hosts) defined in a `[computer]` section of the configuration file.

- `HostName`

Introduced	4.1.3	
Restart Type	system	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a hostname, either this parameter or `ExecuteOnComputer` is required.

If no `HostName` or `ExecuteOnComputer` is specified in a given `[mysql]` or `[api]` section of the `config.ini` file, then an SQL or API node may connect using the corresponding "slot" from any host which can establish a network connection to the management server host machine. *This differs from the default behavior for data nodes, where `localhost` is assumed for `HostName` unless otherwise specified*.

- `ArbitrationRank`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 2`

This parameter defines which nodes can act as arbitrators. Both MGM nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

- `ArbitrationDelay`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	0
	Range	0 .. 4G

Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests will be delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- BatchByteSize

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	32K
	Range	1024 .. 1M

For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (BatchSize) and in terms of bytes (BatchByteSize). The actual batch size is limited by both parameters.

The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set. In future releases, MySQL Server will make educated guesses on how to set parameters relating to batch size, based on the query type.

This parameter is measured in bytes and by default is equal to 32KB.

- BatchSize

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	64
	Range	1 .. 992

This parameter is measured in number of records and is by default set to 64. The maximum size is 992.

- MaxScanBatchSize

Introduced	4.1.3	

Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`256K`
	Range	`32K .. 16M`

The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

You can obtain some information from a MySQL server running as a Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----------------------------+---------------+
| Variable_name               | Value         |
+-----------------------------+---------------+
| Ndb_cluster_node_id         | 5             |
| Ndb_config_from_host        | 192.168.0.112 |
| Ndb_config_from_port        | 1186          |
| Ndb_number_of_storage_nodes | 4             |
+-----------------------------+---------------+
4 rows in set (0.02 sec)
```

For information about these Cluster system status variables, see Section 5.1.5, “Server Status Variables”.

Note

To add new SQL or API nodes to the configuration of a running MySQL Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

15.3.2.7 MySQL Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for all connections between nodes in a MySQL Cluster. Normally it is not necessary to define TCP/IP connections; MySQL Cluster automatically sets up such connections for all data nodes, management nodes, and SQL or API nodes.

Note

For an exception to this rule, see Section 15.3.2.8, “MySQL Cluster TCP/IP Connections Using Direct Connections”.

To override the default connection parameters, it is necessary to define a connection using one or more `[tcp]` sections in the `config.ini` file. Each `[tcp]` section explicitly defines a TCP/IP connection between two MySQL Cluster nodes, and must contain at a minimum the parameters `NodeId1` and `NodeId2`, as well as any connection parameters to override.

It is also possible to change the default values for these parameters by setting them in the `[tcp default]` section.

Important

Any `[tcp]` sections in the `config.ini` file should be listed *last*, following all other sections in the file. However, this is not required for a `[tcp default]` section. This requirement is a known issue with the way in which the `config.ini` file is read by the MySQL Cluster management server.

Connection parameters which can be set in `[tcp]` and `[tcp default]` sections of the `config.ini` file are listed here:

- `NodeId1`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`..`

`NodeId2`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`..`

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file. These are the same unique `Id` values for each of these nodes as described in Section 15.3.2.6, “Defining SQL and Other API Nodes in a MySQL Cluster”.

- `HostName1`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

`HostName2`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	string
	Default	
	Range	..

The HostName1 and HostName2 parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be hostnames or IP addresses.

- SendBufferMemory

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	256K
	Range	64K .. 4G

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer.

The default size of the send buffer is 256 KB; 2MB is recommended in most situations in which it is necessary to set this parameter. The minimum size is 64 KB; the theoretical maximum is 4 GB.

- SendSignalId

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	boolean
	Default	false (debug builds: true)
	Valid Values	true
		false

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to Y, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in -debug builds.

- Checksum

Introduced	4.1.3

Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`boolean`
	Default	`false`
	Valid Values	`true`
		`false`

This parameter is a boolean parameter (enabled by setting it to `Y` or `1`, disabled by setting it to `N` or `0`). It is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- `PortNumber` (*OBSOLETE*)

  This formerly specified the port number to be used for listening for connections from other nodes. This parameter should no longer be used.

- `ReceiveBufferMemory`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`64K`
	Range	`16K .. 4G`

  Specifies the size of the buffer used when receiving data from the TCP/IP socket.

  The default value of this parameter from its of 64 KB; 1M is recommended in most situations where the size of the receive buffer needs to be set. The minimum possible value is 16K; theoretical maximum is 4G.

15.3.2.8 MySQL Cluster TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the `[tcp]` section of the cluster `config.ini` file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the `172.23.72.*` subnet of a LAN. In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the `1.1.0.*` address range as shown:

```
# Management Server
[ndb_mgmd]
Id=1
HostName=172.23.72.20

# SQL Node
[mysqld]
```

```
Id=2
HostName=172.23.72.21

# Data Nodes
[ndbd]
Id=3
HostName=172.23.72.22

[ndbd]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[tcp]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

The `HostName1` [1283] and `HostName2` [1283] parameters are used only when specifying direct connections.

The use of direct TCP connections between data nodes can improve the cluster's overall efficiency by enabling the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency. It is important to note that to take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.

15.3.2.9 MySQL Cluster Shared-Memory Connections

Beginning with MySQL 4.1.9-max, MySQL Cluster attempts to use the shared memory transporter and configure it automatically where possible. (In previous versions of MySQL Cluster, shared memory segments functioned only when the `-max` binary was built using `--with-ndb-shm`.) `[shm]` sections in the `config.ini` file explicitly define shared-memory connections between nodes in the cluster. When explicitly defining shared memory as the connection method, it is necessary to define at least `NodeId1`, `NodeId2` and `ShmKey`. All other parameters have default values that should work well in most cases.

Important

SHM functionality is considered experimental only. It is not officially supported in any current MySQL Cluster release, and testing results indicate that SHM performance is not appreciably greater than when using TCP/IP for the transporter.

For these reasons, you must determine for yourself or by using our free resources (forums, mailing lists) whether SHM can be made to work correctly in your specific case.

-
- `HostName1`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

`HostName2`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`string`
	Default	
	Range	`..`

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be hostnames or IP addresses.

- `ShmKey`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`0 .. 4G`

When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value.

- `ShmSize`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`1M`
	Range	`64K .. 4G`

Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by `ShmSize`. The default value is 1MB.

- `SendSignalId`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`boolean`

	Default	`false`
	Valid Values	`true`
		`false`

To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to `Y` causes these message IDs to be transported over the network as well. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `Checksum`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`boolean`
	Default	`true`
	Valid Values	`true`
		`false`

This parameter is a boolean (`Y`/`N`) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

- `SigNum`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`0 .. 4G`

When using the shared memory transporter, a process sends an operating system signal to the other process when there is new data available in the shared memory. Should that signal conflict with with an existing signal, this parameter can be used to change it. This is a possibility when using SHM due to the fact that different operating systems use different signal numbers.

The default value of `SigNum` is 0; therefore, it must be set to avoid errors in the cluster log when using the shared memory transporter. Typically, this parameter is set to 10 in the `[shm default]` section of the `config.ini` file.

15.3.2.10 SCI Transport Connections in MySQL Cluster

`[sci]` sections in the `config.ini` file explicitly define SCI (Scalable Coherent Interface) connections between cluster nodes. Using SCI transporters in MySQL Cluster is supported only when the MySQL-Max binaries are built using `--with-ndb-sci=/your/path/to/SCI`. The *`path`* should point to a directory

that contains at a minimum `lib` and `include` directories containing SISCI libraries and header files. (See Section 15.3.5, “Using High-Speed Interconnects with MySQL Cluster” for more information about SCI.)

In addition, SCI requires specialized hardware.

It is strongly recommended to use SCI Transporters only for communication between `ndbd` processes. Note also that using SCI Transporters means that the `ndbd` processes never sleep. For this reason, SCI Transporters should be used only on machines having at least two CPUs dedicated for use by `ndbd` processes. There should be at least one CPU per `ndbd` process, with at least one CPU left in reserve to handle operating system activities.

-
- `Host1SciId0`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`0 .. 4G`

This identifies the SCI node ID on the first Cluster node (identified by `NodeId1`).

- `Host1SciId1`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 4G`

It is possible to set up SCI Transporters for failover between two SCI cards which then should use separate networks between the nodes. This identifies the node ID and the second SCI card to be used on the first node.

- `Host2SciId0`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	
	Range	`0 .. 4G`

This identifies the SCI node ID on the second Cluster node (identified by `NodeId2`).

- `Host2SciId1`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	0
	Range	0 .. 4G

When using two SCI cards to provide failover, this parameter identifies the second SCI card to be used on the second node.

- `HostName1`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	string
	Default	
	Range	..

`HostName2`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	string
	Default	
	Range	..

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SCI connection between two nodes. The values used for these parameters can be hostnames or IP addresses.

- `SharedBufferSize`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	numeric
	Default	10M
	Range	64K .. 4G

Each SCI transporter has a shared memory segment used for communication between the two nodes. Setting the size of this segment to the default value of 1MB should be sufficient for most applications.

Using a smaller value can lead to problems when performing many parallel inserts; if the shared buffer is too small, this can also result in a crash of the `ndbd` process.

- `SendLimit`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`numeric`
	Default	`8K`
	Range	`128 .. 32K`

A small buffer in front of the SCI media stores messages before transmitting them over the SCI network. By default, this is set to 8KB. Our benchmarks show that performance is best at 64KB but 16KB reaches within a few percent of this, and there was little if any advantage to increasing it beyond 8KB.

- `SendSignalId`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values	
	Type	`boolean`
	Default	`true`
	Valid Values	`true`
		`false`

To trace a distributed message it is necessary to identify each message uniquely. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `Checksum`

Introduced	4.1.3	
Restart Type	node	
	Permitted Values (>= 4.1.3)	
	Type	`boolean`
	Default	`false`
	Valid Values	`true`
		`false`

This parameter is a boolean value, and is disabled by default. When `Checksum` is enabled, checksums are calculated for all messages before they are placed in the send buffer. This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

15.3.2.11 Configuring MySQL Cluster Parameters for Local Checkpoints

The parameters discussed in Logging and Checkpointing and in Data Memory, Index Memory, and String Memory [1251] that are used to configure local checkpoints for a MySQL Cluster do not exist in isolation, but rather are very much interdepedent on each other. In this section, we illustrate how these parameters—including `DataMemory`, `IndexMemory`, `NoOfDiskPagesToDiskAfterRestartTUP`, `NoOfDiskPagesToDiskAfterRestartACC`, and `NoOfFragmentLogFiles`—relate to one another in a working Cluster.

In this example, we assume that our application performs the following numbers of types of operations per hour:

- 50000 selects
- 15000 inserts
- 15000 updates
- 15000 deletes

We also make the following assumptions about the data used in the application:

- We are working with a single table having 40 columns.
- Each column can hold up to 32 bytes of data.
- A typical `UPDATE` run by the application affects the values of 5 columns.
- No `NULL` values are inserted by the application.

A good starting point is to determine the amount of time that should elapse between local checkpoints (LCPs). It is worth noting that, in the event of a system restart, it takes 40-60 percent of this interval to execute the REDO log—for example, if the time between LCPs is 5 minutes (300 seconds), then it should take 2 to 3 minutes (120 to 180 seconds) for the REDO log to be read.

The maximum amount of data per node can be assumed to be the size of the `DataMemory` parameter. In this example, we assume that this is 2 GB. The `NoOfDiskPagesToDiskAfterRestartTUP` parameter represents the amount of data to be checkpointed per unit time—however, this parameter is actually expressed as the number of 8K memory pages to be checkpointed per 100 milliseconds. 2 GB per 300 seconds is approximately 6.8 MB per second, or 700 KB per 100 milliseconds, which works out to roughly 85 pages per 100 milliseconds.

Similarly, we can calculate `NoOfDiskPagesToDiskAfterRestartACC` in terms of the time for local checkpoints and the amount of memory required for indexes—that is, the `IndexMemory`. Assuming that we permit 512 MB for indexes, this works out to approximately 20 8-KB pages per 100 milliseconds for this parameter.

Next, we need to determine the number of REDO log files required—that is, fragment log files—the corresponding parameter being `NoOfFragmentLogFiles`. We need to make sure that there are sufficient REDO log files for keeping records for at least 3 local checkpoints. In a production setting, there are always uncertainties—for instance, we cannot be sure that disks always operate at top speed or with maximum throughput. For this reason, it is best to err on the side of caution, so we double our requirement and calculate a number of fragment log files which should be enough to keep records covering 6 local checkpoints.

It is also important to remember that the disk also handles writes to the REDO log and UNDO log, so if you find that the amount of data being written to disk as determined by the values of `NoOfDiskPagesToDiskAfterRestartACC` and `NoOfDiskPagesToDiskAfterRestartTUP` is approaching the amount of disk bandwidth available, you may wish to increase the time between local checkpoints.

Given 5 minutes (300 seconds) per local checkpoint, this means that we need to support writing log records at maximum speed for 6 * 300 = 1800 seconds. The size of a REDO log record is 72 bytes plus 4 bytes per updated column value plus the maximum size of the updated column, and there is one REDO log record for each table record updated in a transaction, on each node where the data reside. Using the numbers of operations set out previously in this section, we derive the following:

- 50000 select operations per hour yields 0 log records (and thus 0 bytes), since `SELECT` statements are not recorded in the REDO log.
- 15000 `DELETE` statements per hour is approximately 5 delete operations per second. (Since we wish to be conservative in our estimate, we round up here and in the following calculations.) No columns are updated by deletes, so these statements consume only 5 operations * 72 bytes per operation = 360 bytes per second.
- 15000 `UPDATE` statements per hour is roughly the same as 5 updates per second. Each update uses 72 bytes, plus 4 bytes per column * 5 columns updated, plus 32 bytes per column * 5 columns—this works out to 72 + 20 + 160 = 252 bytes per operation, and multiplying this by 5 operation per second yields 1260 bytes per second.
- 15000 `INSERT` statements per hour is equivalent to 5 insert operations per second. Each insert requires REDO log space of 72 bytes, plus 4 bytes per record * 40 columns, plus 32 bytes per column * 40 columns, which is 72 + 160 + 1280 = 1512 bytes per operation. This times 5 operations per second yields 7560 bytes per second.

So the total number of REDO log bytes being written per second is approximately 0 + 360 + 1260 + 7560 = 9180 bytes. Multiplied by 1800 seconds, this yields 16524000 bytes required for REDO logging, or approximately 15.75 MB. The unit used for `NoOfFragmentLogFiles` represents a set of 4 16-MB log files—that is, 64 MB. Thus, the minimum value (3) for this parameter is sufficient for the scenario envisioned in this example, since 3 times 64 = 192 MB, or about 12 times what is required; the default value of 8 (or 512 MB) is more than ample in this case.

A copy of each altered table record is kept in the UNDO log. In the scenario discussed above, the UNDO log would not require any more space than what is provided by the default seetings. However, given the size of disks, it is sensible to allocate at least 1 GB for it.

15.3.3 Overview of MySQL Cluster Configuration Parameters

The next four sections provide summary tables of MySQL Cluster configuration parameters used in the `config.ini` file to govern the cluster's functioning. Each table lists the parameters for one of the Cluster node process types (`ndbd`, `ndb_mgmd`, and `mysqld`), and includes the parameter's type as well as its default, mimimum, and maximum values as applicable.

These tables also indicate what type of restart is required (node restart or system restart)—and whether the restart must be done with `--initial`—to change the value of a given configuration parameter.

When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a *rolling restart*). It is possible to update cluster configuration parameters marked as `node` online—that is, without shutting down the cluster—in this fashion. An initial node restart requires restarting each `ndbd` process with the `--initial` option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the cluster, wiping the cluster file system after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted for them to read the updated configuration parameter values.

Important

Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. Many of these can be increased online, using a rolling restart.

However, decreasing the values of such parameters—whether this is done using a node restart, node initial restart, or even a complete system restart of the cluster—is not to be undertaken lightly; it is recommended that you do so only after careful planning and testing. This is especially true with regard to those parameters that relate to memory usage and disk space, such as `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes`. In addition, it is the generally the case that configuration parameters relating to memory and disk usage can be raised using a simple node restart, but they require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.

Note

`4294967039`—which often appears as a maximum value in these tables—is defined in the `NDBCLUSTER` sources as `MAX_INT_RNIL` and is equal to `0xFFFFFEFF`, or $2^{32} - 2^{8} - 1$.

15.3.3.1 MySQL Cluster Data Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[ndbd]` or `[ndbd default]` sections of a `config.ini` file for configuring MySQL Cluster data nodes. For detailed descriptions and other additional information about each of these parameters, see Section 15.3.2.5, "Defining MySQL Cluster Data Nodes".

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- `N`—Node restart: The parameter can be updated using a rolling restart (see Section 15.2.6.1, "Performing a Rolling Restart of a MySQL Cluster").
- `S`—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- `I`—Initial restart: Data nodes must be restarted using the `--initial` [1309] option.

For more information about restart types, see Section 15.3.3, "Overview of MySQL Cluster Configuration Parameters".

Table 15.1 Data Node Configuration Parameters

Name	Type/Units	Default	Min Value	Max Value	Restart Type
ArbitrationTimeout [1272]	milliseconds	3000	10	4G	N
BackupDataBufferSize [1277]	bytes	2M		4G	N
BackupDataDir [1251]	path	FileSystemPath			IN
BackupLogBufferSize [1277]	bytes	2M		4G	N
BackupMaxWriteSize [1278]	bytes	256K	2K	4G	N

Name	Type/Units	Default	Min Value	Max Value	Restart Type
BackupMemory [1278]	bytes	4M		4G	N
BackupWriteSize [1278]	bytes	32K	2K	4G	N
BatchSizePerLocalScan [1259]	integer	64	1	992	N
DataDir [1251]	path	.			IN
DataMemory [1252]	bytes	80M	1M	1024G	N
Diskless [1265]	true|false (1|0)	false			IS
ExecuteOnComputer [1249]	name				S
HeartbeatIntervalDbApi [1267]	milliseconds	1500	100	4G	N
HeartbeatIntervalDbDb [1267]	milliseconds	1500	10	4G	N
HostName [1249]	name or IP address	localhost			S
Id [1248]	unsigned		1	48	IS
IndexMemory [1253]	bytes	18M	1M	1T	N
LockPagesInMainMemory [1264]	true|false (1|0)			1	N
LogLevelCheckpoint [1275]	log level			15	N
LogLevelConnection [1276]	integer			15	N
LogLevelError [1276]	integer			15	N
LogLevelInfo [1276]	integer			15	N
LogLevelNodeRestart [1276]	integer			15	N
LogLevelShutdown [1276]	integer			15	N
LogLevelStartup [1274]	integer	1		15	N
LogLevelStatistic [1275]	integer			15	N
LongMessageBuffer [1258]	bytes	1M	512K	4G	N
MaxNoOfAttributes [1261]	integer	1000	32	4G	N
MaxNoOfConcurrentIndexOperations [1257]	integer	8K		4G	N
MaxNoOfConcurrentOperations [1256]	integer	32K	32	4G	N
MaxNoOfConcurrentScans [1258]	integer	256	2	500	N
MaxNoOfConcurrentTransactions [1255]	integer	4096	32	4G	N
MaxNoOfFiredTriggers [1257]	integer	4000		4G	N
MaxNoOfLocalOperations [1257]	integer	UNDEFINED	32	4G	N
MaxNoOfLocalScans [1259]	integer	MaxNoOfConcurrentScans * [# of data nodes] + 2	32	4G	N
MaxNoOfOpenFiles [1261]	integer	40	20	4G	N
MaxNoOfOrderedIndexes [1262]	integer	128		4G	N
MaxNoOfSavedMessages [1261]	integer	25		4G	N
MaxNoOfTables [1262]	integer	128	8	1600	N
MaxNoOfTriggers [1263]	integer	768		4G	N
MaxNoOfUniqueHashIndexes [1263]	integer	64		4G	N

Name	Type/Units	Default	Min Value	Max Value	Restart Type
NoOfDiskPagesToDiskAfterRestartACC [1270]	8K pages/100 milliseconds	20	1	4G	N
NoOfDiskPagesToDiskAfterRestartTUP [1270]	8K pages/100 milliseconds	40	1	4G	N
NoOfDiskPagesToDiskDuringRestartACC [1271]	8K pages/100 milliseconds	20	1	4G	N
NoOfDiskPagesToDiskDuringRestartTUP [1271]	8K pages/100 milliseconds	40	1	4G	N
NoOfFragmentLogFiles [1260]	integer	8	3	4G	IN
NoOfReplicas [1249]	integer		1	4	IS
RedoBuffer [1273]	bytes	8M	1M	4G	N
RestartOnErrorInsert [1265]	error code	2		4	N
ServerPort [1249]	unsigned		1	64K	N
StartFailureTimeout [1267]	milliseconds			4G	N
StartPartialTimeout [1268]	milliseconds	30000		4G	N
StartPartitionedTimeout [1268]	milliseconds	60000		4G	N
StopOnError [1264]		true			N
StringMemory [1253]	% or bytes			4G	S
TimeBetweenGlobalCheckpoints [1268]	milliseconds	2000	10	32000	N
TimeBetweenInactiveTransactionAbortCheck [1269]	milliseconds	1000	1000	4G	N
TimeBetweenLocalCheckpoints [1268]	number of 4-byte words, as a base-2 logarithm	20		31	N
TimeBetweenWatchDogCheck [1266]	milliseconds	6000	70	4G	N
TransactionBufferMemory [1258]	bytes	1M	1K	4G	N
TransactionDeadlockDetectionTimeout [1269]	milliseconds	1200	50	4G	N
TransactionInactiveTimeout [1268]	milliseconds	4G		4G	N
UndoDataBuffer [1273]	unsigned	16M	1M	4G	N
UndoIndexBuffer [1272]	unsigned	2M	1M	4G	N

Note

To add new data nodes to a MySQL Cluster, it is necessary to shut down the cluster completely, update the `config.ini` file, and then restart the cluster, starting all data node processes using the `--initial` [1309] option—that is, you must perform a system restart.

It is possible to add new data node groups to a running cluster online using MySQL Cluster NDB 7.0 or later (see Adding MySQL Cluster Data Nodes Online); however, we do not plan to implement this change in MySQL 4.1.

15.3.3.2 MySQL Cluster Management Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[ndb_mgmd]` or `[mgm]` sections of a `config.ini` file for configuring MySQL Cluster management nodes. For detailed

descriptions and other additional information about each of these parameters, see Section 15.3.2.4, “Defining a MySQL Cluster Management Server”.

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- N—Node restart: The parameter can be updated using a rolling restart (see Section 15.2.6.1, “Performing a Rolling Restart of a MySQL Cluster”).
- S—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- I—Initial restart: Data nodes must be restarted using the `--initial` [1309] option.

For more information about restart types, see Section 15.3.3, “Overview of MySQL Cluster Configuration Parameters”.

Table 15.2 Management Node Configuration Parameters

Name	Type/Units	Default	Min Value	Max Value	Restart Type
ArbitrationDelay [1247]	milliseconds			4G	N
ArbitrationRank [1247]	0-2	1		2	N
DataDir [1247]	path	.			N
ExecuteOnComputer [1245]	name				S
HostName [1245]	name or IP address				S
Id [1244]	unsigned		1	63	IS
LogDestination [1246]	{CONSOLE|SYSLOG|FILE}	FILE: filename=ndb_nodeid_cluster.log, maxsize=1000000, maxfiles=6			N
MaxNoOfSavedEvents	unsigned	100		4G	N
PortNumber [1245]	unsigned	2200		64K	N
PortNumberStats	unsigned			64K	N

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect. See Section 15.3.2.4, “Defining a MySQL Cluster Management Server”, for more information.

To add new management servers to a running MySQL Cluster, it is also necessary perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see Section 15.1.4.9, “Limitations Relating to Multiple MySQL Cluster Nodes”.

15.3.3.3 MySQL Cluster SQL Node and API Node Configuration Parameters

The summary table in this section provides information about parameters used in the `[SQL]` and `[api]` sections of a `config.ini` file for configuring MySQL Cluster SQL nodes and API nodes. For detailed

descriptions and other additional information about each of these parameters, see Section 15.3.2.6, "Defining SQL and Other API Nodes in a MySQL Cluster".

Note

For a discussion of MySQL server options for MySQL Cluster, see Section 15.3.4.2, "`mysqld` Command Options for MySQL Cluster"; for information about MySQL server system variables relating to MySQL Cluster, see Section 15.3.4.3, "MySQL Cluster System Variables".

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary table as follows:

- `N`—Node restart: The parameter can be updated using a rolling restart (see Section 15.2.6.1, "Performing a Rolling Restart of a MySQL Cluster").
- `S`—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- `I`—Initial restart: Data nodes must be restarted using the `--initial` [1309] option.

For more information about restart types, see Section 15.3.3, "Overview of MySQL Cluster Configuration Parameters".

Table 15.3 API Node Configuration Parameters

Name	Type/Units	Default	Min Value	Max Value	Restart Type
ArbitrationDelay [1280]	milliseconds			4G	N
ArbitrationRank [1280]	0-2			2	N
BatchByteSize [1281]	bytes	32K	1024	1M	N
BatchSize [1281]	records	64	1	992	N
ExecuteOnComputer [1280]	name				S
HostName [1280]	name or IP address				S
Id [1279]	unsigned		1	63	IS
MaxScanBatchSize [1281]	bytes	256K	32K	16M	N

Note

To add new SQL or API nodes to the configuration of a running MySQL Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

15.3.3.4 Other MySQL Cluster Configuration Parameters

The summary tables in this section provide information about parameters used in the `[computer]`, `[tcp]`, `[shm]`, and `[sci]` sections of a `config.ini` file for configuring MySQL Cluster management nodes. For detailed descriptions and other additional information about individual parameters, see

Section 15.3.2.7, “MySQL Cluster TCP/IP Connections”, Section 15.3.2.9, “MySQL Cluster Shared-Memory Connections”, or Section 15.3.2.10, “SCI Transport Connections in MySQL Cluster”, as appropriate.

Restart types. Changes in MySQL Cluster configuration parameters do not take effect until the cluster is restarted. The type of restart required to change a given parameter is indicated in the summary tables as follows:

- N—Node restart: The parameter can be updated using a rolling restart (see Section 15.2.6.1, “Performing a Rolling Restart of a MySQL Cluster”).
- S—System restart: The cluster must be shut down completely, then restarted, to effect a change in this parameter.
- I—Initial restart: Data nodes must be restarted using the `--initial` [1309] option.

For more information about restart types, see Section 15.3.3, “Overview of MySQL Cluster Configuration Parameters”.

Table 15.4 COMPUTER Configuration Parameters

Name	Type/Units	Default	Min Value	Max Value	Restart Type
HostName [1244]	name or IP address				S
Id [1244]	string				IS

Table 15.5 TCP Configuration Parameters

Name	Type/Units	Default	Min Value	Max Value	Restart Type
Checksum [1284]		false			N
Group	unsigned	55		200	N
NodeId1 [1283]					N
NodeId2 [1283]					N
NodeIdServer					N
PortNumber [1285]	unsigned			64K	N
Proxy					N
ReceiveBufferMemory [1285]	bytes	64K	16K	4G	N
SendBufferMemory [1284]	unsigned	256K	64K	4G	N
SendSignalId [1284]		false (debug builds: true)			N

Table 15.6 SHM Configuration Parameters

Name	Type/Units	Default	Min Value	Max Value	Restart Type
Checksum [1288]		true			N
Group	unsigned	35		200	N
NodeId1 [ ]					N
NodeId2 [ ]					N
NodeIdServer					N
PortNumber	unsigned			64K	N
SendSignalId [1287]		false			N

Name	Type/Units	Default	Min Value	Max Value	Restart Type
ShmKey [1287]	unsigned			4G	N
ShmSize [1287]	bytes	1M	64K	4G	N
Signum [1288]	unsigned			4G	N

Table 15.7 SCI Configuration Parameters

Name	Type/Units	Default	Min Value	Max Value	Restart Type
Checksum [1291]		false			N
Group	unsigned	15		200	N
Host1SciId0 [1289]	unsigned			4G	N
Host1SciId1 [1289]	unsigned			4G	N
Host2SciId0 [1289]	unsigned			4G	N
Host2SciId1 [1289]	unsigned			4G	N
NodeId1 [ ]					N
NodeId2 [ ]					N
NodeIdServer					N
PortNumber	unsigned			64K	N
SendLimit [1291]	unsigned	8K	128	32K	N
SendSignalId [1291]		true			N
SharedBufferSize [1290]	unsigned	10M	64K	4G	N

15.3.4 MySQL Server Options and Variables for MySQL Cluster

This section provides information about MySQL server options, server and status variables that are specific to MySQL Cluster. For general information on using these, and for other options and variables not specific to MySQL Cluster, see Section 5.1, “The MySQL Server”.

For MySQL Cluster configuration parameters used in the cluster confiuration file (usually named `config.ini`), see Section 15.3, “MySQL Cluster Configuration”.

15.3.4.1 MySQL Cluster Server Option and Variable Reference

The following table provides a list of the command-line options, server and status variables applicable within `mysqld` when it is running as an SQL node in a MySQL Cluster. For a table showing *all* command-line options, server and status variables available for use with `mysqld`, see Section 5.1.1, “Server Option and Variable Reference”.

Table 15.8 MySQL Cluster Server Options and Variables

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_show_ndb_status [450]				Yes	Both	No
Handler_discover [1306]				Yes	Both	No
have_ndbcluster [1302]			Yes		Global	No
ndb_autoincrement_prefetch_sz [1302]	Yes	Yes	Yes		Both	Yes
ndb_cache_check_time [1302]	Yes	Yes	Yes		Global	Yes
ndb_force_send [1302]	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
ndb_index_stat_cache_entries [1303]	Yes	Yes				
ndb_index_stat_enable [1303]	Yes	Yes				
ndb_index_stat_update_freq [1304]	Yes	Yes				
ndb_optimized_node_selection [1304]	Yes	Yes				
ndb_report_thresh_binlog_epoch_slip [1304]	Yes	Yes				
ndb_report_thresh_binlog_mem_usage [1305]	Yes	Yes				
ndb_use_exact_count [1305]			Yes		Both	Yes
ndb_use_transactions [1305]	Yes	Yes	Yes		Both	Yes
ndbcluster [1301]	Yes	Yes				
- *Variable*: have_ndbcluster [1302]						

15.3.4.2 `mysqld` Command Options for MySQL Cluster

This section provides descriptions of `mysqld` server options relating to MySQL Cluster. For information about `mysqld` options not specific to MySQL Cluster, and for general information about the use of options with `mysqld`, see Section 5.1.2, “Server Command Options”.

For information about command-line options used with other MySQL Cluster processes (`ndbd`, `ndb_mgmd`, and `ndb_mgm`), see Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”. For information about command-line options used with `NDB` utility programs (such as `ndb_desc`, `ndb_size.pl`, and `ndb_show_tables`), see Section 15.4, “MySQL Cluster Programs”.

- `--ndb-connectstring=`*`connect_string`*

Command-Line Format	`--ndb-connectstring`	
Option-File Format	`ndb-connectstring`	
	Permitted Values	
	Type	`string`

When using the `NDBCLUSTER` storage engine, this option specifies the management server that distributes cluster configuration data. See Section 15.3.2.2, “The MySQL Cluster Connectstring”, for syntax.

- `--ndbcluster` [1301]

Command-Line Format	`--ndbcluster`	
Option-File Format	`ndbcluster`	
Disabled by	`skip-ndbcluster`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

The `NDBCLUSTER` storage engine is necessary for using MySQL Cluster. If a `mysqld` binary includes support for the `NDBCLUSTER` storage engine, the engine is disabled by default. Use the `--ndbcluster` [1301] option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.

- `--skip-ndbcluster`

Command-Line Format	`--skip-ndbcluster`
Option-File Format	`skip-ndbcluster`

Disable the `NDBCLUSTER` storage engine. This is the default for binaries that were built with `NDBCLUSTER` storage engine support; the server allocates memory and other resources for this storage engine only if the `--ndbcluster` [1301] option is given explicitly. See Section 15.3.1, "Quick Test Setup of MySQL Cluster", for an example.

15.3.4.3 MySQL Cluster System Variables

This section provides detailed information about MySQL server system variables that are specific to MySQL Cluster and the `NDB` storage engine. For system variables not specific to MySQL Cluster, see Section 5.1.3, "Server System Variables". For general information on using system variables, see Section 5.1.4, "Using System Variables".

- `have_ndbcluster` [1302]

Introduced	4.1.2	
System Variable Name	`have_ndbcluster [1302]`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`

`YES` if `mysqld` supports `NDBCLUSTER` tables. `DISABLED` if `--skip-ndbcluster` is used.

- `ndb_autoincrement_prefetch_sz` [1302]

Introduced	4.1.8	
Command-Line Format	`--ndb_autoincrement_prefetch_sz`	
Option-File Format	`ndb_autoincrement_prefetch_sz`	
System Variable Name	`ndb_autoincrement_prefetch_sz [1302]`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`32`
	Range	`1 .. 256`

Determines the probability of gaps in an autoincremented column. Set it to `1` to minimize this. Setting it to a high value for optimization—makes inserts faster, but decreases the likelihood that consecutive autoincrement numbers will be used in a batch of inserts. Default value: `32`. Minimum value: `1`.

- `ndb_cache_check_time` [1302]

Command-Line Format	`--ndb_cache_check_time`
Option-File Format	`ndb_cache_check_time`

System Variable Name	ndb_cache_check_time [1302]	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0

The number of milliseconds that elapse between checks of MySQL Cluster SQL nodes by the MySQL query cache. Setting this to 0 (the default and minimum value) means that the query cache checks for validation on every query.

The recommended maximum value for this variable is 1000, which means that the check is performed once per second. A larger value means that the check is performed and possibly invalidated due to updates on different SQL nodes less often. It is generally not desirable to set this to a value greater than 2000.

- ndb_force_send [1303]

Introduced	4.1.8	
Command-Line Format	--ndb-force-send	
Option-File Format	ndb_force_send	
System Variable Name	ndb_force_send [1303]	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	TRUE

Forces sending of buffers to NDB immediately, without waiting for other threads. Defaults to ON.

- ndb_index_stat_cache_entries [1303]

Command-Line Format	--ndb_index_stat_cache_entries	
Option-File Format	ndb_index_stat_cache_entries	
	Permitted Values	
	Type	numeric
	Default	32
	Range	0 .. 4294967295

Sets the granularity of the statistics by determining the number of starting and ending keys to store in the statistics memory cache. Zero means no caching takes place; in this case, the data nodes are always queried directly. Default value: 32.

- ndb_index_stat_enable [1303]

Command-Line Format	--ndb_index_stat_enable
Option-File Format	ndb_index_stat_enable

	Permitted Values	
	Type	`boolean`
	Default	`ON`

Use `NDB` index statistics in query optimization. Defaults to `ON`.

- `ndb_index_stat_update_freq` [1304]

Command-Line Format	`--ndb_index_stat_update_freq`	
Option-File Format	`ndb_index_stat_update_freq`	
	Permitted Values	
	Type	`numeric`
	Default	`20`
	Range	`0 .. 4294967295`

How often to query data nodes instead of the statistics cache. For example, a value of `20` (the default) means to direct every 20th query to the data nodes.

- `ndb_optimized_node_selection` [1304]

Introduced	4.1.9	
Option-File Format	`ndb_optimized_node_selection`	
	Permitted Values (>= 4.1.9)	
	Type	`boolean`
	Default	`ON`
	Permitted Values	
	Type	`numeric`
	Default	`3`
	Range	`0 .. 3`

Causes an SQL node to use the “closest” data node as transaction coordinator. For this purpose, a data node having a shared memory connection with the SQL node is considered to be “closest” to the SQL node; the next closest (in order of decreasing proximity) are: TCP connection to `localhost`; SCI connection; TCP connection from a host other than `localhost`.

This option is enabled by default. Set to `0` or `OFF` to disable it, in which case the SQL node uses each data node in the cluster in succession. When this option is disabled each SQL thread attempts to use a given data node 8 times before proceeding to the next one.

Added in MySQL 4.1.9.

- `ndb_report_thresh_binlog_epoch_slip` [1304]

Command-Line Format	`--ndb_report_thresh_binlog_epoch_slip`	
Option-File Format	`ndb_report_thresh_binlog_epoch_slip`	
	Permitted Values	
	Type	`numeric`

	Default	`3`
	Range	`0 .. 256`

This is a threshold on the number of epochs to be behind before reporting binlog status. For example, a value of `3` (the default) means that if the difference between which epoch has been received from the storage nodes and which epoch has been applied to the binlog is 3 or more, a status message will be sent to the cluster log.

- `ndb_report_thresh_binlog_mem_usage` [1305]

Command-Line Format	`--ndb_report_thresh_binlog_mem_usage`	
Option-File Format	`ndb_report_thresh_binlog_mem_usage`	
	Permitted Values	
	Type	`numeric`
	Default	`10`
	Range	`0 .. 10`

This is a threshold on the percentage of free memory remaining before reporting binlog status. For example, a value of `10` (the default) means that if the amount of available memory for receiving binlog data from the data nodes falls below 10%, a status message will be sent to the cluster log.

- `ndb_use_exact_count` [1305]

Introduced	4.1.8	
System Variable Name	`ndb_use_exact_count` [1305]	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`ON`

Forces `NDB` to use a count of records during `SELECT COUNT(*)` query planning to speed up this type of query. The default value is `ON`. For faster queries overall, disable this feature by setting the value of `ndb_use_exact_count` [1305] to `OFF`.

- `ndb_use_transactions` [1305]

Introduced	4.1.18	
Command-Line Format	`--ndb_use_transactions`	
Option-File Format	`ndb_use_transactions`	
System Variable Name	`ndb_use_transactions` [1305]	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`ON`

You can disable NDB transaction support by setting this variable's values to OFF (not recommended). The default is ON.

15.3.4.4 MySQL Cluster Status Variables

This section provides detailed information about MySQL server status variables that relate to MySQL Cluster and the NDB storage engine. For status variables not specific to MySQL Cluster, and for general information on using status variables, see Section 5.1.5, “Server Status Variables”.

- Handler_discover [1306]

  The MySQL server can ask the NDBCLUSTER storage engine if it knows about a table with a given name. This is called discovery. Handler_discover [1306] indicates the number of times that tables have been discovered using this mechanism.

  This variable was added in MySQL 4.1.2.

15.3.5 Using High-Speed Interconnects with MySQL Cluster

Even before design of NDBCLUSTER began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, NDBCLUSTER was designed from the very beginning to permit the use of a number of different data transport mechanisms. In this Manual, we use the term *transporter* for these.

The MySQL Cluster codebase includes support for four different transporters:

- *TCP/IP using 100 Mbps or gigabit Ethernet*, as discussed in Section 15.3.2.7, “MySQL Cluster TCP/IP Connections”.
- *Direct (machine-to-machine) TCP/IP*; although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for MySQL Cluster. See Section 15.3.2.8, “MySQL Cluster TCP/IP Connections Using Direct Connections”, for details.
- *Shared memory (SHM)*. For more information about SHM, see Section 15.3.2.9, “MySQL Cluster Shared-Memory Connections”.

  Note

  SHM is considered experimental only, and is not officially supported.

- *Scalable Coherent Interface (SCI)*, as described in the next section of this chapter, Section 15.3.2.10, “SCI Transport Connections in MySQL Cluster”.

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with MySQL Cluster.

We are working to make sure that communication with the ndbd process is made in “chunks” that are as large as possible because this benefits all types of data transmission.

For users who desire it, it is also possible to use cluster interconnects to enhance performance even further. There are two ways to achieve this: Either a custom transporter can be designed to handle this case, or you can use socket implementations that bypass the TCP/IP stack to one extent or another. We have experimented with both of these techniques using the SCI (Scalable Coherent Interface) technology developed by Dolphin.

15.3.5.1 Configuring MySQL Cluster to use SCI Sockets

It is possible employing Scalable Coherent Interface (SCI) technology to achieve a significant increase in connection speeds and throughput between MySQL Cluster data and SQL nodes. To use SCI, it is necessary to obtain and install Dolphin SCI network cards and to use the drivers and other software supplied by Dolphin. You can get information on obtaining these, from Dolphin Interconnect Solutions. SCI SuperSocket or SCI Transporter support is available for 32-bit and 64-bit Linux, Solaris, and other platforms. See the Dolphin documentation referenced later in this section for more detailed information regarding platforms supported for SCI.

Note

Prior to MySQL 4.1.24, there were issues with building MySQL Cluster with SCI support (see Bug #25470), but these have been resolved due to work contributed by Dolphin. SCI Sockets are now correctly supported for MySQL Cluster hosts running recent versions of Linux using the `-max` builds, and versions of MySQL Cluster with SCI Transporter support can be built using either of `compile-amd64-max-sci` or `compile-pentium64-max-sci`. Both of these build scripts can be found in the `BUILD` directory of the MySQL Cluster source trees; it should not be difficult to adapt them for other platforms. Generally, all that is necessary is to compile MySQL Cluster with SCI Transporter support is to configure the MySQL Cluster build using `--with-ndb-sci=/opt/DIS`.

Once you have acquired the required Dolphin hardware and software, you can obtain detailed information on how to adapt a MySQL Cluster configured for normal TCP/IP communication to use SCI from the *Dolphin Express for MySQL Installation and Reference Guide*, available for download at http://docsrva.mysql.com/public/DIS_install_guide_book.pdf (PDF file, 94 pages, 753 KB). This document provides instructions for installing the SCI hardware and software, as well as information concerning network topology and configuration.

15.3.5.2 MySQL Cluster Interconnects and Performance

The `ndbd` process has a number of simple constructs which are used to access the data in a MySQL Cluster. We have created a very simple benchmark to check the performance of each of these and the effects which various interconnects have on their performance.

There are four access methods:

- **Primary key access.** This is access of a record through its primary key. In the simplest case, only one record is accessed at a time, which means that the full cost of setting up a number of TCP/IP messages and a number of costs for context switching are borne by this single request. In the case where multiple primary key accesses are sent in one batch, those accesses share the cost of setting up the necessary TCP/IP messages and context switches. If the TCP/IP messages are for different destinations, additional TCP/IP messages need to be set up.

- **Unique key access.** Unique key accesses are similar to primary key accesses, except that a unique key access is executed as a read on an index table followed by a primary key access on the table. However, only one request is sent from the MySQL Server, and the read of the index table is handled by `ndbd`. Such requests also benefit from batching.

- **Full table scan.** When no indexes exist for a lookup on a table, a full table scan is performed. This is sent as a single request to the `ndbd` process, which then divides the table scan into a set of parallel scans on all cluster `ndbd` processes. In future versions of MySQL Cluster, an SQL node will be able to filter some of these scans.

- **Range scan using ordered index**

When an ordered index is used, it performs a scan in the same manner as the full table scan, except that it scans only those records which are in the range used by the query transmitted by the MySQL server (SQL node). All partitions are scanned in parallel when all bound index attributes include all attributes in the partitioning key.

With benchmarks developed internally by MySQL for testing simple and batched primary and unique key accesses, we have found that using SCI sockets improves performance by approximately 100% over TCP/IP, except in rare instances when communication performance is not an issue. This can occur when scan filters make up most of processing time or when very large batches of primary key accesses are achieved. In that case, the CPU processing in the `ndbd` processes becomes a fairly large part of the overhead.

Using the SCI transporter instead of SCI Sockets is only of interest in communicating between `ndbd` processes. Using the SCI transporter is also only of interest if a CPU can be dedicated to the `ndbd` process because the SCI transporter ensures that this process will never go to sleep. It is also important to ensure that the `ndbd` process priority is set in such a way that the process does not lose priority due to running for an extended period of time, as can be done by locking processes to CPUs in Linux 2.6. If such a configuration is possible, the `ndbd` process will benefit by 10–70% as compared with using SCI sockets. (The larger figures will be seen when performing updates and probably on parallel scan operations as well.)

There are several other optimized socket implementations for computer clusters, including Myrinet, Gigabit Ethernet, Infiniband and the VIA interface. However, we have tested MySQL Cluster so far only with SCI sockets. See Section 15.3.5.1, “Configuring MySQL Cluster to use SCI Sockets”, for information on how to set up SCI sockets using ordinary TCP/IP for MySQL Cluster.

15.4 MySQL Cluster Programs

Using and managing a MySQL Cluster requires several specialized programs, which we describe in this chapter. We discuss the purposes of these programs in a MySQL Cluster, how to use the programs, and what startup options are available for each of them.

These programs include the MySQL Cluster data, management, and SQL node processes (`ndbd`, `ndb_mgmd`, and `mysqld`) and the management client (`ndb_mgm`).

Other NDB utility, diagnostic, and example programs are included with the MySQL Cluster distribution. These include `ndb_restore`, `ndb_show_tables`, and `ndb_config`. These programs are covered later in this chapter.

The last two sections of this chapter contain tables of options used, respectively, with `mysqld` and with the various NDB programs.

15.4.1 ndbd — The MySQL Cluster Data Node Daemon

`ndbd` is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In a MySQL Cluster, a set of `ndbd` processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

The following table includes command options specific to the MySQL Cluster data node program `ndbd`. Additional descriptions follow the table. For options common to all MySQL Cluster programs, see

Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 15.9 ndbd Command Line Options

Format	Description
--daemon [1309]	Start ndbd as daemon (default); override with --nodaemon
--foreground	Run ndbd in foreground, provided for debugging purposes (implies --nodaemon)
--initial [1309]	Perform initial start of ndbd, including cleaning the file system. Consult the documentation before using this option
--nodaemon [1310]	Do not start ndbd as daemon; provided for testing purposes
--nostart [ ]	Don't start ndbd immediately; ndbd waits for command to start from ndb_mgmd

- `--daemon`, `-d`

Command-Line Format	`--daemon`	
	`-d`	
	Permitted Values (>= 4.1.5)	
	Type	`boolean`
	Default	`TRUE`

Instructs `ndbd` to execute as a daemon process. From MySQL 4.1.5 on, this is the default behavior, and `--nodaemon` can be used to prevent the process from running as a daemon.

- `--initial`

Command-Line Format	`--initial`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Instructs `ndbd` to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of `ndbd`. It also re-creates recovery log files. Note that on some operating systems this process can take a substantial amount of time.

An `--initial` start is to be used *only* when starting the `ndbd` process under very special circumstances; this is because this option causes all files to be removed from the Cluster file system and all redo log files to be re-created. These circumstances are listed here:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of `ndbd`.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the data files.

Use of this option prevents the StartPartialTimeout and StartPartitionedTimeout configuration parameters from having any effect.

Important

This option does *not* affect any backup files that have already been created by the affected node.

This option also has no effect on recovery of data by a data node that is just starting (or restarting) from data nodes that are already running. This recovery of data occurs automatically, and requires no user intervention in a MySQL Cluster that is running normally.

In older versions of MySQL Cluster, it was possible to use -i for this option. This shortcut was removed to prevent this option from being used by mistake.

It is permissible to use this option when starting the cluster for the very first time (that is, before any data node files have been created); however, it is *not* necessary to do so.

- --nodaemon

Command-Line Format	--nodaemon	
	Permitted Values	
	Type	boolean
	Default	FALSE

Instructs ndbd not to start as a daemon process. This is useful when ndbd is being debugged and you want output to be redirected to the screen.

-

In MySQL versions prior to 4.1.5, each ndbd process should be started in a separate directory, the reason for this being that ndbd generated a set of log files in its starting directory. In MySQL 4.1.5, this behavior was changed such that these files are placed in the directory specified by DataDir in the configuration file. Thus ndbd can be started from anywhere.

These log files are listed below. *node_id* is the node's unique identifier. Note that *node_id* represents the node's unique identifier. For example, ndb_2_error.log is the error log generated by the data node whose node ID is 2.

-

-

- ndb_*node_id*_trace.log.next (was NextTraceFileNo.log in version 4.1.3) is the file that keeps track of the next trace file number to be assigned.
- ndb_*node_id*_out.log is a file containing any data output by the ndbd process. This file is created only if ndbd is started as a daemon, which is the default behavior beginning with MySQL 4.1.5. This file was named node*node_id*.out in versions 4.1.3 and 4.1.4.
- ndb_*node_id*.pid is a file containing the process ID of the ndbd process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.
- ndb_*node_id*_signal.log (was Signal.log in version 4.1.3) is a file used only in debug versions of ndbd, where it is possible to trace all incoming, outgoing, and internal messages with their data in the ndbd process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the `.pid` file remains in effect even after the process has terminated.

To start `ndbd`, it may also be necessary to specify the host name of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See Section 15.3.2.2, “The MySQL Cluster Connectstring”, for additional information about this issue. Section 15.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”, describes other options for `ndbd`.

When `ndbd` starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the `ndbd` process if it is configured to do so. Thus, if you attempt to kill `ndbd` using the Unix `kill` command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an `ndbd` process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent actions. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the `ndbd` process. In a multi-processor system performing a large number of operations (including updates), the `ndbd` process can consume up to 2 CPUs if permitted to do so.

For a machine with many CPUs it is possible to use several `ndbd` processes which belong to different node groups; however, such a configuration is still considered experimental and is not supported for MySQL 4.1 in a production setting. See Section 15.1.4, “Known Limitations of MySQL Cluster”.

15.4.2 `ndb_mgmd` — The MySQL Cluster Management Server Daemon

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

The following table includes command options specific to the MySQL Cluster management server program `ndb_mgmd`. Additional descriptions follow the table. For options common to all MySQL Cluster programs, see Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 15.10 `ndb_mgmd` Command Line Options

Format	Description	Introduced
-f [ ]	Specify the cluster configuration file; in NDB-6.4.0 and later, needs --reload or --initial to override configuration cache if present	
--daemon [ ]	Run ndb_mgmd in daemon mode (default)	
--interactive	Run ndb_mgmd in interactive mode (not officially supported in production; for testing purposes only)	
--mycnf	Read cluster configuration data from the my.cnf file	
--no-nodeid-checks	Do not provide any node id checks	
--nodaemon [1312]	Do not run ndb_mgmd as a daemon	
--print-full-config [ ]	Print full configuration and exit	4.1.14

-
-
- `--nodaemon`

Command-Line Format	`--nodaemon`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

  Instructs `ndb_mgmd` not to start as a daemon process.

-

As of MySQL 4.1.5, it is no longer necessary to specify a connectstring when starting the management server. However, if you are using more than one management server, a connectstring should be provided and each node in the cluster should specify its node ID explicitly.

See Section 15.3.2.2, “The MySQL Cluster Connectstring”, for information about using connectstrings. Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”, describes other options for `ndb_mgmd`.

The following files are created or used by `ndb_mgmd` in its starting directory. From MySQL 4.1.5, the log and PID files are placed in the `DataDir` as specified in the `config.ini` configuration file. In the list that follows, *`node_id`* is the unique node identifier.

-

- `ndb_`*`node_id`*`_cluster.log` (was `cluster.log` in version 4.1.3) is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in Section 15.5, “Management of MySQL Cluster”.

  When the size of the cluster log reaches one million bytes, the file is renamed to `ndb_`*`node_id`*`_cluster.log.`*`seq_id`* (was `cluster.log.`*`seq_id`* in version 4.1.3) where *`seq_id`* is the sequence number of the cluster log file. (For example: If files with the sequence numbers 1, 2, and 3 already exist, the next log file is named using the number `4`.)

- `ndb_`*`node_id`*`_out.log` (was `node1.out` in version 4.1.3) is the file used for `stdout` and `stderr` when running the management server as a daemon.

- `ndb_`*`node_id`*`.pid` (was `node`*`node_id`*`.pid` in version 4.1.3) is the process ID file used when running the management server as a daemon.

15.4.3 ndb_mgm — The MySQL Cluster Management Client

The `ndb_mgm` management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by `ndb_mgm`.

To start the management client, it is necessary to supply the host name and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```

For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default host name and port number are `localhost` and 1186, respectively. (Prior to MySQL 4.1.8, the default Cluster port was 2200.)

The following table includes command options specific to the MySQL Cluster management client program `ndb_mgm`. Additional descriptions follow the table. For options common to all MySQL Cluster programs, see Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 15.11 `ndb_mgm` Command Line Options

Format	Description
--execute=name [ ]	Execute command and exit
--try-reconnect=# [1313]	Specify number of tries for connecting to ndb_mgmd (0 = infinite)

-
- `--try-reconnect=`*`number`*

Command-Line Format	`--try-reconnect=#`	
	`-t`	
	Permitted Values	
	Type	`boolean`
	Default	`TRUE`

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to *`number`* before giving up and reporting an error instead.

Additional information about using `ndb_mgm` can be found in Section 15.5.2, “Commands in the MySQL Cluster Management Client”.

15.4.4 `ndb_config` — Extract MySQL Cluster Configuration Information

This tool extracts configuration information for data nodes, SQL nodes, and API nodes from a cluster management node (and possibly its `config.ini` file).

The following table includes command options specific to the MySQL Cluster program `ndb_config`. Additional descriptions follow the table. For options common to all MySQL Cluster programs, see Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 15.12 `ndb_config` Command Line Options

Format	Description
--config-file=path [1314]	Set the path to config.ini file
--fields=string [1316]	Field separator
--host=name [1315]	Specify host
--mycnf	Read configuration data from my.cnf file
--nodeid	Get configuration of node with this ID
--nodes [1315]	Print node information (DB section) only.
--query=string [1314]	One or more query options (attributes)

Format	Description
--rows=string [1316]	Row separator
--type=name [1315]	Specify node type

- `--usage`, `--help`, or `-?`

Command-Line Format	`--help`
	`--usage`
	`-?`

Causes `ndb_config` to print a list of available options, and then exit.

- `--version`, `-V`

Command-Line Format	`--version`
	`-V`

Causes `ndb_config` to print a version information string, and then exit.

- `--ndb-connectstring=connect_string`

Command-Line Format	`--ndb-connectstring=connectstring`	
	`--connect-string=connectstring`	
	`-c`	
	Permitted Values	
	Type	`string`
	Default	`localhost:1186`

Specifies the connectstring to use in connecting to the management server. The format for the connectstring is the same as described in Section 15.3.2.2, “The MySQL Cluster Connectstring”, and defaults to `localhost:1186`.

The use of `-c` as a short version for this option is not currently supported with `ndb_config`.

- `--config-file=path-to-file`

Command-Line Format	`--config-file=path`	
	Permitted Values	
	Type	`file name`
	Default	

Gives the path to the management server's configuration file (`config.ini`). This may be a relative or absolute path. If the management node resides on a different host from the one on which `ndb_config` is invoked, then an absolute path must be used.

- `--query=query-options`, `-q query-options`

Command-Line Format	`--query=string`
	`-q`

	Permitted Values	
	Type	string
	Default	

This is a comma-delimited list of *query options*—that is, a list of one or more node attributes to be returned. These include `id` (node ID), type (node type—that is, `ndbd`, `mysqld`, or `ndb_mgmd`), and any configuration parameters whose values are to be obtained.

For example, `--query=id,type,indexmemory,datamemory` would return the node ID, node type, `DataMemory`, and `IndexMemory` for each node.

Note

If a given parameter is not applicable to a certain type of node, than an empty string is returned for the corresponding value. See the examples later in this section for more information.

- `--host=hostname`

Command-Line Format	`--host=name`	
	Permitted Values	
	Type	string
	Default	

Specifies the host name of the node for which configuration information is to be obtained.

- `--id=node_id`, `--nodeid=node_id`

Command-Line Format	`--ndb-nodeid=#`	
	Permitted Values	
	Type	numeric
	Default	0

Used to specify the node ID of the node for which configuration information is to be obtained.

- `--nodes`

Command-Line Format	`--nodes`	
	Permitted Values	
	Type	boolean
	Default	FALSE

(Tells `ndb_config` to print information from parameters defined in `[ndbd]` sections only. Currently, using this option has no affect, since these are the only values checked, but it may become possible in future to query parameters set in `[tcp]` and other sections of cluster configuration files.)

- `--type=node_type`

Command-Line Format	`--type=name`	
	Permitted Values	

	Type	enumeration
	Default	
	Valid Values	ndbd
		mysqld
		ndb_mgmd

Filters results so that only configuration values applying to nodes of the specified *node_type* (ndbd, mysqld, or ndb_mgmd) are returned.

- --fields=*delimiter*, -f *delimiter*

Command-Line Format	--fields=string	
	-f	
	Permitted Values	
	Type	string
	Default	

Specifies a *delimiter* string used to separate the fields in the result. The default is "," (the comma character).

Note

If the *delimiter* contains spaces or escapes (such as \n for the linefeed character), then it must be quoted.

- --rows=*separator*, -r *separator*

Command-Line Format	--rows=string	
	-r	
	Permitted Values	
	Type	string
	Default	

Specifies a *separator* string used to separate the rows in the result. The default is a space character.

Note

If the *separator* contains spaces or escapes (such as \n for the linefeed character), then it must be quoted.

Examples:

1. To obtain the node ID and type of each node in the cluster:

```
shell> ./ndb_config --query=id,type --fields=':' --rows='\n'
1:ndbd
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysqld
7:mysqld
```

```
8:mysqld
9:mysqld
```

In this example, we used the `--fields` options to separate the ID and type of each node with a colon character (`:`), and the `--rows` options to place the values for each node on a new line in the output.

2. To produce a connectstring that can be used by data, SQL, and API nodes to connect to the management server:

```
shell> ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini --query=hostname,portnumber -
192.168.0.179:1186
```

3. This invocation of `ndb_config` checks only data nodes (using the `--type` option), and shows the values for each node's ID and host name, and its `DataMemory`, `IndexMemory`, and `DataDir` parameters:

```
shell> ./ndb_config --type=ndbd --query=id,host,datamemory,indexmemory,datadir -f ' : ' -r '\n'
1 : 192.168.0.193 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
2 : 192.168.0.112 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
3 : 192.168.0.176 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
4 : 192.168.0.119 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
```

In this example, we used the short options `-f` and `-r` for setting the field delimiter and row separator, respectively.

4. To exclude results from any host except one in particular, use the `--host` option:

```
shell> ./ndb_config --host=192.168.0.176 -f : -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

In this example, we also used the short form `-q` to determine the attributes to be queried.

Similarly, you can limit results to a node with a specific ID using the `--id` or `--nodeid` option.

15.4.5 ndb_cpcd — Automate Testing for NDB Development

This utility is found in the `libexec` directory. It is part of an internal automated test framework used in testing and debugging MySQL Cluster. Because it can control processes on remote systems, it is not advisable to use `ndb_cpcd` in a production cluster.

The source files for `ndb_cpcd` may be found in the directory `ndb/src/cw/cpcd`, in the MySQL 4.1 source tree.

15.4.6 ndb_delete_all — Delete All Rows from an NDB Table

`ndb_delete_all` deletes all rows from the given `NDB` table. In some cases, this can be much faster than `DELETE` or even `TRUNCATE TABLE`.

Usage:

```
ndb_delete_all -c connect_string tbl_name -d db_name
```

This deletes all rows from the table named *`tbl_name`* in the database named *`db_name`*. It is exactly equivalent to executing `TRUNCATE db_name.tbl_name` in MySQL.

Additional Options:

- `--transactional`, `-t`

  Use of this option causes the delete operation to be performed as a single transaction.

Warning

With very large tables, using this option may cause the number of operations available to the cluster to be exceeded.

15.4.7 ndb_desc — Describe NDB Tables

`ndb_desc` provides a detailed description of one or more `NDB` tables.

Usage:

```
ndb_desc -c connect_string tbl_name -d db_name [-p]
```

Sample Output:

MySQL table creation and population statements:

```
USE test;

CREATE TABLE fish (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(20),

    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
) ENGINE=NDBCLUSTER;

INSERT INTO fish VALUES
    ('','guppy'), ('','tuna'), ('','shark'),
    ('','manta ray'), ('','grouper'), ('','puffer');
```

Output from `ndb_desc`:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 16777221
Fragment type: 5
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 2
Number of primary keys: 1
Length of frm data: 268
Row Checksum: 1
Row GCI: 1
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
name Varchar(20;latin1_swedish_ci) NULL AT=SHORT_VAR ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
uk(name) - OrderedIndex
PRIMARY(id) - OrderedIndex
```

```
uk$unique(name) - UniqueHashIndex

-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory  Frag varsized memory
2          2          2             65536              327680
1          2          2             65536              327680
3          2          2             65536              327680

NDBT_ProgramExit: 0 - OK
```

Additional Options:

- `--extra-partition-info`, `-p`

  Prints additional information about the table's partitions.

- Information about multiple tables can be obtained in a single invocation of `ndb_desc` by using their names, separated by spaces. All of the tables must be in the same database.

15.4.8 ndb_drop_index — Drop Index from an NDB Table

`ndb_drop_index` drops the specified index from an `NDB` table. *It is recommended that you use this utility only as an example for writing NDB API applications*—see the Warning later in this section for details.

Usage:

```
ndb_drop_index -c connect_string table_name index -d db_name
```

The statement shown above drops the index named *`index`* from the *`table`* in the *`database`*.

Additional Options: None that are specific to this application.

Warning

Operations performed on Cluster table indexes using the NDB API are not visible to MySQL and make the table unusable by a MySQL server. If you use this program to drop an index, then try to access the table from an SQL node, an error results, as shown here:

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *******
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.1.25

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+------------------+
| Tables_in_ctest1 |
+------------------+
| a                |
| bt1              |
```

```
| bt2               |
| dogs              |
| employees         |
| fish              |
+-------------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

In such a case, your *only* option for making the table available to MySQL again is to drop the table and re-create it. You can use either the SQL statementDROP TABLE or the ndb_drop_table utility (see Section 15.4.9, “ndb_drop_table — Drop an NDB Table”) to drop the table.

15.4.9 ndb_drop_table — Drop an NDB Table

ndb_drop_table drops the specified NDB table. (If you try to use this on a table created with a storage engine other than NDB, it fails with the error 723: No such table exists.) This operation is extremely fast—in some cases, it can be an order of magnitude faster than using DROP TABLE on an NDB table from MySQL.

Usage:

```
ndb_drop_table -c connect_string tbl_name -d db_name
```

Additional Options: None.

15.4.10 ndb_error_reporter — NDB Error-Reporting Utility

ndb_error_reporter creates an archive from data node and management node log files that can be used to help diagnose bugs or other problems with a cluster. *It is highly recommended that you make use of this utility when filing reports of bugs in MySQL Cluster*.

The following table includes command options specific to the MySQL Cluster program ndb_error_reporter. Additional descriptions follow the table. For options common to all MySQL Cluster programs, see Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 15.13 ndb_error_reporter Command Line Options

Format	Description
--fs [1320]	Include file system data in error report; can use a large amount of disk space

Usage:

```
ndb_error_reporter path/to/config-file [username] [--fs]
```

This utility is intended for use on a management node host, and requires the path to the management host configuration file (config.ini). Optionally, you can supply the name of a user that is able to access the cluster's data nodes using SSH, to copy the data node log files. ndb_error_reporter then includes all of these files in archive that is created in the same directory in which it is run. The archive is named ndb_error_report_*YYYYMMDDHHMMSS*.tar.bz2, where *YYYYMMDDHHMMSS* is a datetime string.

If the --fs is used, then the data node file systems are also copied to the management host and included in the archive that is produced by this script. As data node file systems can be extremely large even after

being compressed, we ask that you please do *not* send archives created using this option to Oracle unless you are specifically requested to do so.

Command-Line Format	--fs	
	Permitted Values	
	Type	boolean
	Default	FALSE

15.4.11 ndb_print_backup_file — Print NDB Backup File Contents

ndb_print_backup_file obtains diagnostic information from a cluster backup file.

Usage:

```
ndb_print_backup_file file_name
```

file_name is the name of a cluster backup file. This can be any of the files (.Data, .ctl, or .log file) found in a cluster backup directory. These files are found in the data node's backup directory under the subdirectory BACKUP-#, where # is the sequence number for the backup. For more information about cluster backup files and their contents, see Section 15.5.3.1, "MySQL Cluster Backup Concepts".

Like ndb_print_schema_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_backup_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options: None.

15.4.12 ndb_print_schema_file — Print NDB Schema File Contents

ndb_print_schema_file obtains diagnostic information from a cluster schema file.

Usage:

```
ndb_print_schema_file file_name
```

file_name is the name of a cluster schema file. For more information about cluster schema files, see MySQL Cluster Data Node FileSystemDir Files.

Like ndb_print_backup_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_schema_backup_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options: None.

15.4.13 ndb_print_sys_file — Print NDB System File Contents

ndb_print_sys_file obtains diagnostic information from a MySQL Cluster system file.

Usage:

```
ndb_print_sys_file file_name
```

file_name is the name of a cluster system file (sysfile). Cluster system files are located in a data node's data directory (DataDir); the path under this directory to system files matches the pattern ndb_#_fs/D#/DBDIH/P#.sysfile. In each case, the # represents a number (not necessarily the same number). For more information, see MySQL Cluster Data Node FileSystemDir Files.

Like ndb_print_backup_file and ndb_print_schema_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_backup_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options: None.

15.4.14 ndb_restore — Restore a MySQL Cluster Backup

The cluster restoration program is implemented as a separate command-line utility ndb_restore, which can normally be found in the MySQL bin directory. This program reads the files created as a result of the backup and inserts the stored information into the database.

ndb_restore must be executed once for each of the backup files that were created by the START BACKUP command used to create the backup (see Section 15.5.3.2, "Using The MySQL Cluster Management Client to Create a Backup"). This is equal to the number of data nodes in the cluster at the time that the backup was created.

Note

Before using ndb_restore, it is recommended that the cluster be running in single user mode, unless you are restoring multiple data nodes in parallel. See Section 15.5.7, "MySQL Cluster Single User Mode", for more information about single user mode.

The following table includes command options specific to the MySQL Cluster native backup restoration program ndb_restore. Additional descriptions follow the table. For options common to all MySQL Cluster programs, see Section 15.4.20, "Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs".

Table 15.14 ndb_restore Command Line Options

Format	Description
--backupid=# [1323]	Restore from the backup with the given ID
--connect [1323]	Alias for --connectstring.
--dont_ignore_systab_0	Do not ignore system table during restore. Experimental only; not for production use
--nodeid=# [1323]	Back up files from node with this ID
--parallelism=#	Number of parallel transactions to use while restoring data
--print	Print metadata, data and log to stdout (equivalent to --print_meta --print_data --print_log)
--print_data	Print data to stdout
--print_log	Print to stdout
--print_meta	Print metadata to stdout

Format	Description
--restore_data	Restore table data and logs into NDB Cluster using the NDB API
--restore_meta	Restore metadata to NDB Cluster using the NDB API
--verbose=#	Level of verbosity in output

Typical options for this utility are shown here:

```
ndb_restore [-c connectstring] -n node_id [-m] -b backup_id -r [backup_path=]/path/to/backup/files
```

The `-c` option is used to specify a connectstring which tells `ndb_restore` where to locate the cluster management server. (See Section 15.3.2.2, “The MySQL Cluster Connectstring”, for information on connectstrings.) If this option is not used, then `ndb_restore` attempts to connect to a management server on `localhost:1186`. This utility acts as a cluster API node, and so requires a free connection “slot” to connect to the cluster management server. This means that there must be at least one `[api]` or `[mysqld]` section that can be used by it in the cluster `config.ini` file. It is a good idea to keep at least one empty `[api]` or `[mysqld]` section in `config.ini` that is not being used for a MySQL server or other application for this reason (see Section 15.3.2.6, “Defining SQL and Other API Nodes in a MySQL Cluster”).

You can verify that `ndb_restore` is connected to the cluster by using the `SHOW` command in the `ndb_mgm` management client. You can also accomplish this from a system shell, as shown here:

```
shell> ndb_mgm -e "SHOW"
```

`-n` is used to specify the node ID of the data node on which the backups were taken.

The first time you run the `ndb_restore` restoration program, you also need to restore the metadata. In other words, you must re-create the database tables—this can be done by running it with the `-m` option. Note that the cluster should have an empty database when starting to restore a backup. (In other words, you should start `ndbd` with `--initial` prior to performing the restore.)

The `-b` option is used to specify the ID or sequence number of the backup, and is the same number shown by the management client in the `Backup backup_id completed` message displayed upon completion of a backup. (See Section 15.5.3.2, “Using The MySQL Cluster Management Client to Create a Backup”.)

Important

When restoring cluster backups, you must be sure to restore all data nodes from backups having the same backup ID. Using files from different backups will at best result in restoring the cluster to an inconsistent state, and may fail altogether.

The path to the backup directory is required; this is supplied to `ndb_restore` using the `--backup_path` option, and must include the subdirectory corresponding to the ID backup of the backup to be restored. For example, if the data node's `DataDir` is `/var/lib/mysql-cluster`, then the backup directory is `/var/lib/mysql-cluster/BACKUP`, and the backup files for the backup with the ID 3 can be found in `/var/lib/mysql-cluster/BACKUP/BACKUP-3`. The path may be absolute or relative to the directory in which the `ndb_restore` executable is located, and may be optionally prefixed with `backup_path=`.

Important

It is not possible to restore a backup made from a newer version of MySQL Cluster using an older version of `ndb_restore`. You can restore a backup made from a newer version of MySQL to an older cluster, but you must use a copy of `ndb_restore` from the newer MySQL Cluster version to do so.

> For example, to restore a cluster backup taken from a cluster running MySQL 4.1.22 to a cluster running MySQL Cluster 4.1.20, you must use a copy of ndb_restore from the 4.1.22 distribution.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID 12, created in a cluster with two database nodes having the node IDs 2 and 3, is to be restored to a cluster with four nodes. Then ndb_restore must be run twice—once for each database node in the cluster where the backup was taken. However, ndb_restore cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version. See Section 15.2.6.2, "MySQL Cluster 4.1 Upgrade and Downgrade Compatibility", for more information.

Important

It is not possible to restore a backup made from a newer version of MySQL Cluster using an older version of ndb_restore. You can restore a backup made from a newer version of MySQL to an older cluster, but you must use a copy of ndb_restore from the newer MySQL Cluster version to do so.

For example, to restore a cluster backup taken from a cluster running MySQL 4.1.22 to a cluster running MySQL Cluster 4.1.18, you must use a copy of ndb_restore from the 4.1.22 distribution.

For more rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. That is, when restoring to multiple nodes in parallel, you must have an [api] or [mysqld] section in the cluster config.ini file available for each concurrent ndb_restore process. However, the data files must always be applied before the logs.

Note

If a table has no explicit primary key, then the output generated when using the --print includes the table's hidden primary key.

Error reporting. ndb_restore reports both temporary and permanent errors. In the case of temporary errors, it may able to recover from them. Beginning with MySQL 4.1.22, it reports Restore successful, but encountered temporary error, please look at configuration in such cases.

15.4.15 ndb_select_all — Print Rows from an NDB Table

ndb_select_all prints all rows from an NDB table to stdout.

Usage:

```
ndb_select_all -c connect_string tbl_name -d db_name [> file_name]
```

Additional Options:

- --lock=*lock_type*, -l *lock_type*

  Employs a lock when reading the table. Possible values for *lock_type* are:

  - 0: Read lock
  - 1: Read lock with hold
  - 2: Exclusive read lock

There is no default value for this option.

- `--order=index_name`, `-o index_name`

  Orders the output according to the index named *`index_name`*. Note that this is the name of an index, not of a column, and that the index must have been explicitly named when created.

- `--descending`, `-z`

  Sorts the output in descending order. This option can be used only in conjunction with the `-o` (`--order`) option.

- `--header=FALSE`

  Excludes column headers from the output.

- `--useHexFormat -x`

  Causes all numeric values to be displayed in hexadecimal format. This does not affect the output of numerals contained in strings or datetime values.

- `--delimiter=character`, `-D character`

  Causes the *`character`* to be used as a column delimiter. Only table data columns are separated by this delimiter.

  The default delimiter is the tab character.

- `--rowid`

  Adds a `ROWID` column providing information about the fragments in which rows are stored.

- `--gci`

  Adds a column to the output showing the global checkpoint at which each row was last updated. See Section 15.1, “MySQL Cluster Overview”, and Section 15.5.5.2, “MySQL Cluster Log Events”, for more information about checkpoints.

- `--tupscan`, `-t`

  Scan the table in the order of the tuples.

- `--nodata`

  Causes any table data to be omitted.

Sample Output:

Output from a MySQL `SELECT` statement:

```
mysql> SELECT * FROM ctest1.fish;
+----+-----------+
| id | name      |
+----+-----------+
|  3 | shark     |
|  6 | puffer    |
|  2 | tuna      |
|  4 | manta ray |
|  5 | grouper   |
```

```
|  1 | guppy     |
+----+-----------+
6 rows in set (0.04 sec)
```

Output from the equivalent invocation of ndb_select_all:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id      name
3       [shark]
6       [puffer]
2       [tuna]
4       [manta ray]
5       [grouper]
1       [guppy]
6 rows returned

NDBT_ProgramExit: 0 - OK
```

Note that all string values are enclosed by square brackets ("[...]") in the output of ndb_select_all. For a further example, consider the table created and populated as shown here:

```
CREATE TABLE dogs (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(25) NOT NULL,
    breed VARCHAR(50) NOT NULL,
    PRIMARY KEY pk (id),
    KEY ix (name)
)
ENGINE=NDBCLUSTER;

INSERT INTO dogs VALUES
    ('', 'Lassie', 'collie'),
    ('', 'Scooby-Doo', 'Great Dane'),
    ('', 'Rin-Tin-Tin', 'Alsatian'),
    ('', 'Rosscoe', 'Mutt');
```

This demonstrates the use of several additional ndb_select_all options:

```
shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci
GCI     id name           breed
834461  2  [Scooby-Doo]   [Great Dane]
834878  4  [Rosscoe]      [Mutt]
834463  3  [Rin-Tin-Tin]  [Alsatian]
835657  1  [Lassie]       [Collie]
4 rows returned

NDBT_ProgramExit: 0 - OK
```

15.4.16 ndb_select_count — Print Row Counts for NDB Tables

ndb_select_count prints the number of rows in one or more NDB tables. With a single table, the result is equivalent to that obtained by using the MySQL statement SELECT COUNT(*) FROM *tbl_name*.

Usage:

```
ndb_select_count [-c connect_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

Additional Options: None that are specific to this application. However, you can obtain row counts from multiple tables in the same database by listing the table names separated by spaces when invoking this command, as shown under **Sample Output**.

Sample Output:

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs

NDBT_ProgramExit: 0 - OK
```

15.4.17 ndb_show_tables — Display List of NDB Tables

ndb_show_tables displays a list of all NDB database objects in the cluster. By default, this includes not only both user-created tables and NDB system tables, but NDB-specific indexes, and internal triggers, as well.

The following table includes command options specific to the MySQL Cluster program ndb_show_tables. Additional descriptions follow the table. For options common to all MySQL Cluster programs, see Section 15.4.20, “Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs”.

Table 15.15 ndb_show_tables Command Line Options

Format	Description
--database=string [1327]	Specifies the database in which the table is found
--loops=# [1327]	Number of times to repeat output
--parsable [1327]	Return output suitable for MySQL LOAD DATA INFILE statement
--show-temp-status [1327]	Show table temporary flag
--type=# [1327]	Limit output to objects of this type
--unqualified [1328]	Do not qualify table names

Usage:

```
ndb_show_tables [-c connect_string]
```

Additional Options:

- --database, -d

  Specifies the name of the database in which the tables are found.

- --loops, -l

  Specifies the number of times the utility should execute. This is 1 when this option is not specified, but if you do use the option, you must supply an integer argument for it.

- --parsable, -p

  Using this option causes the output to be in a format suitable for use with LOAD DATA INFILE.

- --show-temp-status

  If specified, this causes temporary tables to be displayed.

- --type, -t

  Can be used to restrict the output to one type of object, specified by an integer type code as shown here:

- **1**: System table
- **2**: User-created table
- **3**: Unique hash index

Any other value causes all NDB database objects to be listed (the default).

- `--unqualified`, `-u`

  If specified, this causes unqualified object names to be displayed.

Note

Only user-created Cluster tables may be accessed from MySQL; system tables such as SYSTAB_0 are not visible to mysqld. However, you can examine the contents of system tables using NDB API applications such as ndb_select_all (see Section 15.4.15, “ndb_select_all — Print Rows from an NDB Table”).

15.4.18 ndb_size.pl — NDBCLUSTER Size Requirement Estimator

This is a Perl script that can be used to estimate the amount of space that would be required by a MySQL database if it were converted to use the NDBCLUSTER storage engine. Unlike the other utilities discussed in this section, it does not require access to a MySQL Cluster (in fact, there is no reason for it to do so). However, it does need to access the MySQL server on which the database to be tested resides.

Requirements:

- A running MySQL server. The server instance does not have to provide support for MySQL Cluster.
- A working installation of Perl.
- The DBI and HTML::Template modules, both of which can be obtained from CPAN if they are not already part of your Perl installation. (Many Linux and other operating system distributions provide their own packages for one or both of these libraries.)
- The ndb_size.tmpl template file, which you should be able to find in the share/mysql directory of your MySQL installation. This file should be copied or moved into the same directory as ndb_size.pl—if it is not there already—before running the script.
- A MySQL user account having the necessary privileges. If you do not wish to use an existing account, then creating one using GRANT USAGE ON *db_name*.*—where *db_name* is the name of the database to be examined—is sufficient for this purpose.

ndb_size.pl and ndb_size.tmpl can also be found in the MySQL sources in storage/ndb/tools.

Usage:

```
perl ndb_size.pl db_name hostname username password > file_name.html
```

The command shown connects to the MySQL server at *hostname* using the account of the user *username* having the password *password*, analyzes all of the tables in database *db_name*, and generates a report in HTML format which is directed to the file *file_name*.html. (Without the redirection, the output is sent to stdout.) This figure shows a portion of the generated ndb_size.html output file, as viewed in a Web browser:

MySQL Cluster analysis for world

This is an automated analysis of the DBI:mysql:database=world;host=192.168.0.176 database for migration into MySQL Cluster. No warranty is made to the accuracy of the information.

This information should be valid for MySQL 4.1 and 5.0. Since 5.1 is not a final release yet, the numbers should be used as a guide only.

Parameter Settings

NOTE the configuration parameters below do not take into account system tables and other requirements.

Parameter	4.1	5.0	5.1
DataMemory (kb)	544	544	544
IndexMemory (kb)	192	136	136
MaxNoOfTables	3	3	3
MaxNoOfAttributes	24	24	24
MaxNoOfOrderedIndexes	3	3	3
MaxNoOfUniqueHashIndexes	3	3	3
MaxNoOfTriggers	12	12	12

Memory usage because of parameters

Usage is in kilobytes. Actual usage will vary as you should set the parameters larger than those listed in the table above.

Parameter	4.1	5.0	5.1
Attributes	5	5	5
Tables	60	60	60
OrderedIndexes	30	30	30
UniqueHashIndexes	45	45	45

Table List

City

Column	Type	Size	Key	4.1 NDB Size	5.0 NDB Size	5.1 NDB Size
ID	int	11	PRI	4	4	4
District	char	20		20	20	20
Name	char	35		36	36	36
Population	int	11		4	4	4
CountryCode	char	3		4	4	4

Indexes

We assume that indexes are ORDERED (not created USING HASH). If order is not required, 10 bytes of data memory can be saved per row if the index is created USING HASH

Index	Type	Columns	4.1 IdxMem	5.0 IdxMem	5.1 IdxMem	4.1 DatMem	5.0 DatMem	5.1 DatMem
PRIMARY	BTREE	ID	29	25	25	10	10	10

DataMemory Usage

	4.1	5.0	5.1
Row Overhead	16	16	16
Column DataMemory/Row	68	68	68
Index DataMemory/Row	10	10	10
Total DataMemory/Row	94	94	94
Rows per 32kb page	347	347	347
Current number of rows	4079	4079	4079
Total DataMemory (kb)	384	384	384

IndexMemory Usage

The output from this script includes:

- Minimum values for the DataMemory, IndexMemory, MaxNoOfTables, MaxNoOfAttributes, MaxNoOfOrderedIndexes, MaxNoOfUniqueHashIndexes, and MaxNoOfTriggers configuration parameters required to accommodate the tables analyzed.
- Memory requirements for all of the tables, attributes, ordered indexes, and unique hash indexes defined in the database.
- The IndexMemory and DataMemory required per table and table row.

15.4.19 ndb_waiter — Wait for MySQL Cluster to Reach a Given Status

ndb_waiter repeatedly (each 100 milliseconds) prints out the status of all cluster data nodes until either the cluster reaches a given status or the --timeout limit is exceeded, then exits. By default, it waits for the cluster to achieve STARTED status, in which all nodes have started and connected to the cluster. This can be overridden using the --no-contact and --not-started options (see Additional Options [1330]).

The node states reported by this utility are as follows:

- NO_CONTACT: The node cannot be contacted.
- UNKNOWN: The node can be contacted, but its status is not yet known. Usually, this means that the node has received a START or RESTART command from the management server, but has not yet acted on it.
- NOT_STARTED: The node has stopped, but remains in contact with the cluster. This is seen when restarting the node using the management client's RESTART command.
- STARTING: The node's ndbd process has started, but the node has not yet joined the cluster.
- STARTED: The node is operational, and has joined the cluster.
- SHUTTING_DOWN: The node is shutting down.
- SINGLE USER MODE: This is shown for all cluster data nodes when the cluster is in single user mode.

Usage:

```
ndb_waiter [-c connect_string]
```

Additional Options:

- --no-contact, -n

  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NO_CONTACT status before exiting.

- --not-started

  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NOT_STARTED status before exiting.

- --timeout=*seconds*, -t *seconds*

  Time to wait. The program exits if the desired state is not achieved within this number of seconds. The default is 120 seconds (1200 reporting cycles).

Sample Output. Shown here is the output from ndb_waiter when run against a 4-node cluster in which two nodes have been shut down and then started again manually. Duplicate reports (indicated by "...") are omitted.

```
shell> ./ndb_waiter -c localhost

Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED

NDBT_ProgramExit: 0 - OK
```

Note

If no connectstring is specified, then `ndb_waiter` tries to connect to a management on `localhost`, and reports `Connecting to mgmsrv at (null)`.

15.4.20 Options Common to MySQL Cluster Programs — Options Common to MySQL Cluster Programs

All MySQL Cluster programs (except for `mysqld`) take the options described in this section as of MySQL 4.1.8. Users of earlier MySQL Cluster versions should note that some of these options have been changed to make consistent with one another as well as with `mysqld`. You can use the `--help` option with any MySQL Cluster program to view a list of the options which it supports.

Table 15.16 Common MySQL Cluster Command line Options

Format	Description
--character-sets-dir=path [ ]	Directory where character sets are
--core-file [1333]	Write core on errors (defaults to TRUE in debug builds)
--debug=options [1333]	Enable output from debug calls. Can be used only for versions compiled with debugging enabled
--help [ ]	Display help message and exit
--ndb-connectstring=connectstring	Set connection string for connecting to ndb_mgmd. Syntax: [nodeid=<id>;][host=]<hostname>[:<port>]. Overrides entries specified in NDB_CONNECTSTRING or my.cnf.
--ndb-mgmd-host=host[:port]	Set the host (and port, if desired) for connecting to the management server
--ndb-nodeid=# [1333]	Set node id for this node
--ndb-optimized-node-selection [1334]	Select nodes for transactions in a more optimal way
--ndb-shm	Allow for optimization using shared memory connections where available (was EXPERIMENTAL, later REMOVED)
--version [ ]	Output version information and exit

For options specific to individual MySQL Cluster programs, see Section 15.4, “MySQL Cluster Programs”.

See Section 15.3.4.2, “`mysqld` Command Options for MySQL Cluster”, for `mysqld` options relating to MySQL Cluster.

-
-
- `--connect-string=connect_string`

Command-Line Format	`--ndb-connectstring=connectstring`	
	`--connect-string=connectstring`	
	`-c`	
	Permitted Values	
	Type	`string`
	Default	`localhost:1186`

connect_string sets the connectstring to the management server as a command option. Available with ndb_mgm beginning with MySQL 4.1.8.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

For more information, see Section 15.3.2.2, “The MySQL Cluster Connectstring”.

- --core-file

Command-Line Format	--core-file	
	Permitted Values	
	Type	boolean
	Default	FALSE

Write a core file if the program dies. The name and location of the core file are system-dependent. (For MySQL Cluster programs nodes running on Linux, the default location is the program's working directory—for a data node, this is the node's DataDir.) For some systems, there may be restrictions or limitations; for example, it might be necessary to execute ulimit -c unlimited before starting the server. Consult your system documentation for detailed information.

If MySQL Cluster was built using the --debug option for configure, then --core-file is enabled by default. For regular builds, --core-file is disabled by default.

- --debug[=*options*]

Command-Line Format	--debug=options	
	Permitted Values	
	Type	string
	Default	d:t:O,/tmp/ndb_restore.trace

This option can be used only for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the mysqld process.

- --ndb-mgmd-host=*host*[:*port*]

Command-Line Format	--ndb-mgmd-host=host[:port]	
Option-File Format	ndb-mgmd-host	
	Permitted Values	
	Type	string
	Default	localhost:1186

Can be used to set the host and port number of the management server to connect to.

- --ndb-nodeid=#

Command-Line Format	--ndb-nodeid=#

	Permitted Values	
	Type	numeric
	Default	0

Sets this node's MySQL Cluster node ID. *The range of permitted values depends on the type of the node (data, management, or API) and the version of the MySQL Cluster software which is running on it*. See Section 15.1.4.2, “Limits and Differences of MySQL Cluster from Standard MySQL Limits”, for more information.

- `--ndb-optimized-node-selection`

Command-Line Format	`--ndb-optimized-node-selection`	
	Permitted Values	
	Type	boolean
	Default	TRUE

Optimize selection of nodes for transactions. Enabled by default.

-

15.5 Management of MySQL Cluster

Managing a MySQL Cluster involves a number of tasks, the first of which is to configure and start MySQL Cluster. This is covered in Section 15.3, “MySQL Cluster Configuration”, and Section 15.4, “MySQL Cluster Programs”.

The next few sections cover the management of a running MySQL Cluster.

For information about security issues relating to management and deployment of a MySQL Cluster, see Section 15.5.9, “MySQL Cluster Security Issues”.

There are essentially two methods of actively managing a running MySQL Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_`*`node_id`*`_cluster.log`; this is usually found in the management server's `DataDir` directory, but this location can be overridden using the `LogDestination` option—see Section 15.3.2.4, “Defining a MySQL Cluster Management Server”, for details. (Recall that *`node_id`* represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by `ndbd`. It is also possible to send cluster log entries to a Unix system log.

In addition, some aspects of the cluster's operation can be monitored from an SQL node using the `SHOW ENGINE NDB STATUS` statement. See Section 12.4.5.9, “`SHOW ENGINE` Syntax”, for more information.

15.5.1 Summary of MySQL Cluster Start Phases

This section provides a simplified outline of the steps involved when MySQL Cluster data nodes are started. More complete information can be found in MySQL Cluster Start Phases.

These phases are the same as those reported in the output from the *`node_id`* `STATUS` command in the management client. (See Section 15.5.2, “Commands in the MySQL Cluster Management Client”, for more information about this command.)

Start types. There are several different startup types and modes, as shown here:

- **Initial Start.** The cluster starts with a clean file system on all data nodes. This occurs either when the cluster started for the very first time, or when all data nodes are restarted using the `--initial` option.
- **System Restart.** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.
- **Node Restart.** This is the online restart of a cluster node while the cluster itself is running.
- **Initial Node Restart.** This is the same as a node restart, except that the node is reinitialized and started with a clean file system.

Setup and initialization (Phase -1). Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a node ID
2. Fetch configuration data
3. Allocate ports to be used for inter-node communications
4. Allocate memory according to settings obtained from the configuration file

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one `ndbd` reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and `ndb_mgmd`.

Normally, in the event of a problem with the node, the node disconnects from the management server, the socket used for the connection is closed, and the reserved node ID is freed. However, if a node is disconnected abruptly—for example, due to a hardware failure in one of the cluster hosts, or because of network issues—the normal closing of the socket by the operating system may not take place. In this case, the node ID continues to be reserved and not released until a TCP timeout occurs 10 or so minutes later.

To take care of this problem, you can use `PURGE STALE SESSIONS`. Running this statement forces all reserved node IDs to be checked; any that are not being used by nodes actually connected to the cluster are then freed.

Beginning with MySQL 5.1.11, timeout handling of node ID assignments is implemented. This performs the ID usage checks automatically after approximately 20 seconds, so that `PURGE STALE SESSIONS` should no longer be necessary in a normal Cluster start.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Phase 0.** The `NDBFS` and `NDBCNTR` blocks start (see `NDB` Kernel Blocks). The cluster file system is cleared, if the cluster was started with the `--initial` option.
- **Phase 1.** In this stage, all remaining `NDB` kernel blocks are started. Cluster connections are set up, inter-block communications are established, and Cluster heartbeats are started. In the case of a node restart, API node connections are also checked.

Note

When one or more nodes hang in Phase 1 while the remaining node or nodes hang in Phase 2, this often indicates network problems. One possible cause

> of such issues is one or more cluster hosts having multiple network interfaces. Another common source of problems causing this condition is the blocking of TCP/IP ports needed for communications between cluster nodes. In the latter case, this is often due to a misconfigured firewall.

- **Phase 2.** The `NDBCNTR` kernel block checks the states of all existing nodes. The master node is chosen, and the cluster schema file is initialized.
- **Phase 3.** The `DBLQH` and `DBTC` kernel blocks set up communications between them. The startup type is determined; if this is a restart, the `DBDIH` block obtains permission to perform the restart.
- **Phase 4.** For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles`.

  For a system restart:

  - Read schema or schemas.
  - Read data from the local checkpoint.
  - Apply all redo information until the latest restorable global checkpoint has been reached.

  For a node restart, find the tail of the redo log.
- **Phase 5.** Most of the database-related portion of a data node start is perfomed during this phase. For an initial start or system restart, a local checkpoint is executed, followed by a global checkpoint. Periodic checks of memory usage begin during this phase, and any required node takeovers are performed.
- **Phase 6.** In this phase, node groups are defined and set up.
- **Phase 7.** The arbitrator node is selected and begins to function. The next backup ID is set, as is the backup disk write speed. Nodes reaching this start phase are marked as `Started`. It is now possible for API nodes (including SQL nodes) to connect to the cluster. connect.
- **Phase 8.** If this is a system restart, all indexes are rebuilt (by `DBDIH`).
- **Phase 9.** The node internal startup variables are reset.
- **Phase 100 (*OBSOLETE*).** Formerly, it was at this point during a node restart or initial node restart that API nodes could connect to the node and begin to receive events. Currently, this phase is empty.
- **Phase 101.** At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly joined node takes over responsibility for delivering its primary data to subscribers. This phase is also referred to as *SUMA handover phase*.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

15.5.2 Commands in the MySQL Cluster Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client `ndb_mgm`. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in Section 15.5.5, “Event Reports Generated in MySQL Cluster”; commands for creating backups and restoring from them are provided in Section 15.5.3, “Online Backup of MySQL Cluster”.

The management client has the following basic commands. In the listing that follows, *`node_id`* denotes either a database node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

-
-
-
-
-
-
-
-
-
-

15.5.3 Online Backup of MySQL Cluster

The next few sections describe how to prepare for and then to create a MySQL Cluster backup using the functionality for this purpose found in the `ndb_mgm` management client. To distinguish this type of backup from a backup made using `mysqldump`, we sometimes refer to it as a “native” MySQL Cluster backup. (For information about the creation of backups with `mysqldump`, see Section 4.5.4, “`mysqldump` — A Database Backup Program”.) Restoration of MySQL Cluster backups is done using the `ndb_restore` utility provided with the MySQL Cluster distribution; for information about `ndb_restore` and its use in restoring MySQL Cluster backups, see Section 15.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”.

15.5.3.1 MySQL Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata.** The names and definitions of all database tables
- **Table records.** The data actually stored in the database tables at the time that the backup was made
- **Transaction log.** A sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.ctl`

  A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- `BACKUP-`*`backup_id`*`-0.`*`node_id`*`.data`

  A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.log`

  A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host

15.5.3.2 Using The MySQL Cluster Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See Section 15.5.3.3, “Configuration for MySQL Cluster Backups”.)

The `START BACKUP` command is used to create a backup:

```
START BACKUP [backup_id] [wait_option]

wait_option:
WAIT {STARTED | COMPLETED} | NOWAIT
```

Successive backups are automatically identified sequentially, so the *`backup_id`*, an integer greater than or equal to 1, is optional; if it is omitted, the next available value is used. If an existing *`backup_id`* value is used, the backup fails with the error `Backup failed: file already exists`. If used, the *`backup_id`* must follow `START BACKUP` immediately, before any other options are used.

The maximum supported value for *`backup_id`* in MySQL 4.1 is 2147483648 (2^{31}). (Bug #43042)

Note

If you start a backup using `ndb_mgm -e "START BACKUP"`, the *`backup_id`* is required.

The *`wait_option`* can be used to determine when control is returned to the management client after a `START BACKUP` command is issued, as shown in the following list:

-
-
-

`WAIT COMPLETED` is the default.

The procedure for creating a backup consists of the following steps:

1. Start the management client (`ndb_mgm`), if it not running already.
2. Execute the **`START BACKUP`** command. This produces several lines of output indicating the progress of the backup, as shown here:

   ```
   ndb_mgm> START BACKUP
   Waiting for completed, this may take several minutes
   Node 2: Backup 1 started from node 1
   Node 2: Backup 1 started from node 1 completed
    StartGCP: 177 StopGCP: 180
    #Records: 7362 #LogRecords: 0
    Data: 453648 bytes Log: 0 bytes
   ndb_mgm>
   ```

3.
4. The management client indicates with a message like this one that the backup has started:

   ```
   Backup backup_id started from node node_id completed
   ```

   As is the case for the notification that the backup has started, *`backup_id`* is the unique identifier for this particular backup, and *`node_id`* is the node ID of the management server that is coordinating the backup with the data nodes. This output is accompanied by additional information including relevant global checkpoints, the number of records backed up, and the size of the data, as shown here:

   ```
   ```

```
Node 2: Backup 1 started from node 1 completed
 StartGCP: 177 StopGCP: 180
 #Records: 7362 #LogRecords: 0
 Data: 453648 bytes Log: 0 bytes
```

It is also possible to perform a backup from the system shell by invoking `ndb_mgm` with the `-e` or `--execute` option, as shown in this example:

```
shell> ndb_mgm -e "START BACKUP 6 WAIT COMPLETED"
```

When using `START BACKUP` in this way, you must specify the backup ID.

Cluster backups are created by default in the `BACKUP` subdirectory of the `DataDir` on each data node. This can be overridden for one or more data nodes individually, or for all cluster data nodes in the `config.ini` file using the `BackupDataDir` configuration parameter as discussed in Identifying Data Nodes [ ]. The backup files created for a backup with a given *`backup_id`* are stored in a subdirectory named `BACKUP-`*`backup_id`* in the backup directory.

To abort a backup that is already in progress:

1. Start the management client.
2. Execute this command:

   ```
   ndb_mgm> ABORT BACKUP backup_id
   ```

   The number *`backup_id`* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup` *`backup_id`* `started from node` *`management_node_id`*).
3. The management client will acknowledge the abort request with `Abort of backup` *`backup_id`* `ordered`.

   Note

   At this point, the management client has not yet received a response from the cluster data nodes to this request, and the backup has not yet actually been aborted.
4. After the backup has been aborted, the management client will report this fact in a manner similar to what is shown here:

   ```
   Node 1: Backup 3 started from 5 has been aborted. Error: 1321 - Backup aborted by user request: Permane
   Node 3: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
   Node 2: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
   Node 4: Backup 3 started from 5 has been aborted. Error: 1323 - 1323: Permanent error: Internal error
   ```

   In this example, we have shown sample output for a cluster with 4 data nodes, where the sequence number of the backup to be aborted is 3, and the management node to which the cluster management client is connected has the node ID 5. The first node to complete its part in aborting the backup reports that the reason for the abort was due to a request by the user. (The remaining nodes report that the backup was aborted due to an unspecified internal error.)

   Note

   There is no guarantee that the cluster nodes respond to an `ABORT BACKUP` command in any particular order.

The `Backup backup_id started from node management_node_id has been aborted` messages mean that the backup has been terminated and that all files relating to this backup have been removed from the cluster file system.

It is also possible to abort a backup in progress from a system shell using this command:

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

Note

If there is no backup having the ID *backup_id* running when an `ABORT BACKUP` is issued, the management client makes no response, nor is it indicated in the cluster log that an invalid abort command was sent.

15.5.3.3 Configuration for MySQL Cluster Backups

Five configuration parameters are essential for backup:

- `BackupDataBufferSize`

  The amount of memory used to buffer data before it is written to disk.

- `BackupLogBufferSize`

  The amount of memory used to buffer log records before these are written to disk.

- `BackupMemory`

  The total memory allocated in a database node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- `BackupWriteSize`

  The default size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

- `BackupMaxWriteSize`

  The maximum size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

More detailed information about these parameters can be found in Backup Parameters [ ].

15.5.3.4 MySQL Cluster Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup.

Important

If you have set `BackupDataBufferSize` and `BackupLogBufferSize` and their sum is greater than 4MB, then you must also set `BackupMemory` as well. See `BackupMemory` [1278].

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

NDB does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring a MySQL Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get nonrepeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

15.5.4 MySQL Server Usage for MySQL Cluster

mysqld is the traditional MySQL server process. To be used with MySQL Cluster, mysqld needs to be built with support for the NDBCLUSTER storage engine, as it is in the precompiled -max binaries available from http://dev.mysql.com/downloads/ for MySQL versions 4.1.3 and newer. If you build MySQL from source, you must invoke configure with the --with-ndbcluster option to enable NDB Cluster storage engine support.

For information about other MySQL server options and variables relevant to MySQL Cluster in addition to those discussed in this section, see Section 15.3.4, “MySQL Server Options and Variables for MySQL Cluster”.

If the mysqld binary has been built with Cluster support, the NDBCLUSTER storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use --ndbcluster [1301] as a startup option on the command line when starting mysqld.
- Insert a line containing NDBCLUSTER in the [mysqld] section of your my.cnf file.

An easy way to verify that your server is running with the NDBCLUSTER storage engine enabled is to issue the SHOW ENGINES statement in the MySQL Monitor (mysql). You should see the value YES as the Support value in the row for NDBCLUSTER. If you see NO in this row or if there is no such row displayed in the output, you are not running an NDB-enabled version of MySQL. If you see DISABLED in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID
- The host name or IP address for the management server (MGM node)
- The number of the TCP/IP port on which it can connect to the management server

Beginning with MySQL 4.1.5, node IDs can be dynamically allocated, in which case there is no need to specify them explicitly.

The mysqld parameter ndb-connectstring is used to specify the connectstring either on the command line when starting mysqld or in my.cnf. The connectstring contains the host name or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, ndb_mgmd.mysql.com is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See Section 15.3.2.2, “The MySQL Cluster Connectstring”, for more information on connectstrings.

Given this information, the MySQL server will be a full participant in the cluster. (We often refer to a mysqld process running in this manner as an SQL node.) It will be fully aware of all cluster data nodes as well as their status, and will establish connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the `mysql` client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS` [492] privilege, then the first row of the output is as shown here:

```
mysql> SHOW PROCESSLIST \G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db:
Command: Daemon
   Time: 1
  State: Waiting for event from ndbcluster
   Info: NULL
```

Important

To participate in a MySQL Cluster, the `mysqld` process must be started with *both* the options `--ndbcluster` [1301] and `--ndb-connectstring` (or their equivalents in `my.cnf`). If `mysqld` is started with only the `--ndbcluster` [1301] option, or if it is unable to contact the cluster, it is not possible to work with `NDB` tables, *nor is it possible to create any new tables regardless of storage engine*. The latter restriction is a safety measure intended to prevent the creation of tables having the same names as `NDB` tables while the SQL node is not connected to the cluster. If you wish to create tables using a different storage engine while the `mysqld` process is not participating in a MySQL Cluster, you must restart the server *without* the `--ndbcluster` [1301] option.

15.5.5 Event Reports Generated in MySQL Cluster

In this section, we discuss the types of event logs provided by MySQL Cluster, and the types of events that are logged.

MySQL Cluster provides two types of event log:

- The *cluster log*, which includes events generated by all cluster nodes. The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single location.

  By default, the cluster log is saved to a file named `ndb_node_id_cluster.log`, (where *`node_id`* is the node ID of the management server) in the same directory where the `ndb_mgm` binary resides.

  Cluster logging information can also be sent to `stdout` or a `syslog` facility in addition to or instead of being saved to a file, as determined by the values set for the `DataDir` and `LogDestination` configuration parameters. See Section 15.3.2.4, “Defining a MySQL Cluster Management Server”, for more information about these parameters.

- *Node logs* are local to each node.

  Output generated by node event logging is written to the file `ndb_node_id_out.log` (where *`node_id`* is the node's node ID) in the node's `DataDir`. Node event logs are generated for both management nodes and data nodes.

  Node logs are intended to be used only during application development, or for debugging application code.

Both types of event logs can be set to log different subsets of events.

Each reportable event can be distinguished according to three different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.
- *Priority*: This is represented by one of the numbers from 1 to 15 inclusive, where 1 indicates “most important” and 15 “least important.”
- *Severity Level*: This can be any one of the following values: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG`.

Both the cluster log and the node log can be filtered on these properties.

The format used in the cluster log is as shown here:

```
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9: API version 5.1.15
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9: API version 5.1.15
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
```

Each line in the cluster log contains the following information:

- A timestamp in *`YYYY-MM-DD HH:MM:SS`* format.
- The type of node which is performing the logging. In the cluster log, this is always `[MgmSrvr]`.
- The severity of the event.
- The ID of the node reporting the event.
- A description of the event. The most common types of events to appear in the log are connections and disconnections between different nodes in the cluster, and when checkpoints occur. In some cases, the description may contain status information.

15.5.5.1 MySQL Cluster Logging Management Commands

The following management commands are related to the cluster log:

- `CLUSTERLOG ON`

  Turns the cluster log on.

- `CLUSTERLOG OFF`

Turns the cluster log off.

- `CLUSTERLOG INFO`

  Provides information about cluster log settings.

- *`node_id`* `CLUSTERLOG` *`category=threshold`*

  Logs *`category`* events with priority less than or equal to *`threshold`* in the cluster log.

- `CLUSTERLOG FILTER` *`severity_level`*

  Toggles cluster logging of events of the specified *`severity_level`*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

Note that events are reported per data node, and that the threshold can be set to different values on different nodes.

Category	Default threshold (All data nodes)
`STARTUP`	**7**
`SHUTDOWN`	**7**
`STATISTICS`	**7**
`CHECKPOINT`	**7**
`NODERESTART`	**7**
`CONNECTION`	**7**
`ERROR`	**15**
`INFO`	**7**

The `STATISTICS` category can provide a great deal of useful data. See Section 15.5.5.3, "Using `CLUSTERLOG STATISTICS` in the MySQL Cluster Management Client", for more information.

Thresholds are used to filter events within each category. For example, a `STARTUP` event with a priority of 3 is not logged unless the threshold for `STARTUP` is set to 3 or higher. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels.

Note

These correspond to Unix `syslog` levels, except for `LOG_EMERG` and `LOG_NOTICE`, which are not used or mapped.

1	`ALERT`	A condition that should be corrected immediately, such as a corrupted system database
2	`CRITICAL`	Critical conditions, such as device errors or insufficient resources
3	`ERROR`	Conditions that should be corrected, such as configuration errors
4	`WARNING`	Conditions that are not errors, but that might require special handling

5	`INFO`	Informational messages
6	`DEBUG`	Debugging messages used for `NDBCLUSTER` development

Event severity levels can be turned on or off (using `CLUSTERLOG FILTER`—see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

Important

Cluster log levels are set on a per `ndb_mgmd`, per subscriber basis. This means that, in a MySQL Cluster with multiple management servers, using a `CLUSTERLOG` command in an instance of `ndb_mgm` connected to one management server affects only logs generated by that management server but not by any of the others. This also means that, should one of the management servers be restarted, only logs generated by that management server are affected by the resetting of log levels caused by the restart.

15.5.5.2 MySQL Cluster Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint,” respectively.

`CONNECTION` Events

These events are associated with connections between Cluster nodes.

Event	Priority	Severity Level	Description
data nodes connected	8	`INFO`	Data nodes connected
data nodes disconnected	8	`INFO`	Data nodes disconnected
Communication closed	8	`INFO`	SQL node or data node connection closed
Communication opened	8	`INFO`	SQL node or data node connection opened

`CHECKPOINT` Events

The logging messages shown here are associated with checkpoints.

Event	Priority	Severity Level	Description
LCP stopped in calc keep GCI	0	`ALERT`	LCP stopped
Local checkpoint fragment completed	11	`INFO`	LCP on a fragment has been completed

Event	Priority	Severity Level	Description
Global checkpoint completed	10	INFO	GCP finished
Global checkpoint started	9	INFO	Start of GCP: REDO log is written to disk
Local checkpoint completed	8	INFO	LCP completed normally
Local checkpoint started	7	INFO	Start of LCP: data written to disk
Report undo log blocked	7	INFO	UNDO logging blocked; buffer near overflow

STARTUP Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Event	Priority	Severity Level	Description
Internal start signal received STTORRY	15	INFO	Blocks received after completion of restart
Undo records executed	15	INFO	
New REDO log started	10	INFO	GCI keep *X*, newest restorable GCI *Y*
New log started	10	INFO	Log part *X*, start MB *Y*, stop MB *Z*
Node has been refused for inclusion in the cluster	8	INFO	Node cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
data node neighbors	8	INFO	Shows neighboring data nodes
data node start phase *X* completed	4	INFO	A data node start phase has been completed
Node has been successfully included into the cluster	3	INFO	Displays the node, managing node, and dynamic ID
data node start phases initiated	1	INFO	NDB Cluster nodes starting
data node all start phases completed	1	INFO	NDB Cluster nodes started
data node shutdown initiated	1	INFO	Shutdown of data node has commenced
data node shutdown aborted	1	INFO	Unable to shut down data node normally

NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Event	Priority	Severity Level	Description
Node failure phase completed	8	ALERT	Reports completion of node failure phases
Node has failed, node state was *X*	8	ALERT	Reports that a node has failed
Report arbitrator results	2	ALERT	There are eight different possible results for arbitration attempts:

Event	Priority	Severity Level	Description
			• Arbitration check failed—less than 1/2 nodes left • Arbitration check succeeded—node group majority • Arbitration check failed—missing node group • Network partitioning—arbitration required • Arbitration succeeded—affirmative response from node *X* • Arbitration failed - negative response from node *X* • Network partitioning - no arbitrator available • Network partitioning - no arbitrator configured
Completed copying a fragment	10	INFO	
Completed copying of dictionary information	8	INFO	
Completed copying distribution information	8	INFO	
Starting to copy fragments	8	INFO	
Completed copying all fragments	8	INFO	
GCP takeover started	7	INFO	
GCP takeover completed	7	INFO	
LCP takeover started	7	INFO	
LCP takeover completed (state = *X*)	7	INFO	
Report whether an arbitrator is found or not	6	INFO	There are seven different possible outcomes when seeking an arbitrator: • Management server restarts arbitration thread [state=*X*] • Prepare arbitrator node *X* [ticket=*Y*] • Receive arbitrator node *X* [ticket=*Y*] • Started arbitrator node *X* [ticket=*Y*] • Lost arbitrator node *X* - process failure [state=*Y*] • Lost arbitrator node *X* - process exit [state=*Y*] • Lost arbitrator node *X* <error msg> [state=*Y*]

`STATISTICS` Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Event	Priority	Severity Level	Description
Report job scheduling statistics	9	`INFO`	Mean internal job scheduling statistics
Sent number of bytes	9	`INFO`	Mean number of bytes sent to node *X*
Received # of bytes	9	`INFO`	Mean number of bytes received from node *X*
Report transaction statistics	8	`INFO`	Numbers of: transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts
Report operations	8	`INFO`	Number of operations
Report table create	7	`INFO`	
Memory usage	5	`INFO`	Data and index memory usage (80%, 90%, and 100%)

`ERROR` Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

Event	Priority	Severity	Description
Dead due to missed heartbeat	8	`ALERT`	Node *X* declared “dead” due to missed heartbeat
Transporter errors	2	ERROR	
Transporter warnings	8	`WARNING`	
Missed heartbeats	8	`WARNING`	Node *X* missed heartbeat #*Y*
General warning events	2	`WARNING`	

`INFO` Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Event	Priority	Severity	Description
Sent heartbeat	12	`INFO`	Heartbeat sent to node *X*
Create log bytes	11	`INFO`	Log part, log file, MB
General information events	2	`INFO`	

15.5.5.3 Using `CLUSTERLOG STATISTICS` in the MySQL Cluster Management Client

The `NDB` management client's `CLUSTERLOG STATISTICS` command can provide a number of useful statistics in its output. Counters providing information about the state of the cluster are updated at 5-second reporting intervals by the transaction coordinator (TC) and the local query handler (LQH), and written to the cluster log.

Transaction coordinator statistics. Each transaction has one transaction coordinator, which is chosen by one of the following methods:

- In a round-robin fashion
- By communication proximity

Note

You can determine which TC selection method is used for transactions started from a given SQL node using the `ndb_optimized_node_selection` [1304] system variable. For more information, see Section 15.3.4.3, “MySQL Cluster System Variables”.

All operations within the same transaction use the same transaction coordinator, which reports the following statistics:

- **`Trans count`.** This is the number transactions started in the last interval using this TC as the transaction coordinator. Any of these transactions may have committed, have been aborted, or remain uncommitted at the end of the reporting interval.

  Note

  Transactions do not migrate between TCs.

- **`Commit count`.** This is the number of transactions using this TC as the transaction coordinator that were committed in the last reporting interval. Because some transactions committed in this reporting interval may have started in a previous reporting interval, it is possible for `Commit count` to be greater than `Trans count`.

- **`Read count`.** This is the number of primary key read operations using this TC as the transaction coordinator that were started in the last reporting interval, including simple reads. This count also includes reads performed as part of unique index operations. A unique index read operation generates 2 primary key read operations—1 for the hidden unique index table, and 1 for the table on which the read takes place.

- **`Simple read count`.** This is the number of simple read operations using this TC as the transaction coordinator that were started in the last reporting interval. This is a subset of `Read count`. Because the value of `Simple read count` is incremented at a different point in time from `Read count`, it can lag behind `Read count` slightly, so it is conceivable that `Simple read count` is not equal to `Read count` for a given reporting interval, even if all reads made during that time were in fact simple reads.

- **`Write count`.** This is the number of primary key write operations using this TC as the transaction coordinator that were started in the last reporting interval. This includes all inserts, updates, writes and deletes, as well as writes performed as part of unique index operations.

  Note

  A unique index update operation can generate multiple PK read and write operations on the index table and on the base table.

- **`AttrInfoCount`.** This is the number of 32-bit data words received in the last reporting interval for primary key operations using this TC as the transaction coordinator. For reads, this is proportional to the number of columns requested. For inserts and updates, this is proportional to the number of columns written, and the size of their data. For delete operations, this is usually zero. Unique index operations generate multiple PK operations and so increase this count. However, data words sent to describe the

PK operation itself, and the key information sent, are *not* counted here. Attribute information sent to describe columns to read for scans, or to describe ScanFilters, is also not counted in `AttrInfoCount`.

- **`Concurrent Operations`.** This is the number of primary key or scan operations using this TC as the transaction coordinator that were started during the last reporting interval but that were not completed. Operations increment this counter when they are started and decrement it when they are completed; this occurs after the transaction commits. Dirty reads and writes—as well as failed operations—decrement this counter. The maximum value that `Concurrent Operations` can have is the maximum number of operations that a TC block can support; currently, this is `(2 * MaxNoOfConcurrentOperations) + 16 + MaxNoOfConcurrentTransactions`. (For more information about these configuration parameters, see the *Transaction Parameters* section of Section 15.3.2.5, “Defining MySQL Cluster Data Nodes”.)

- **`Abort count`.** This is the number of transactions using this TC as the transaction coordinator that were aborted during the last reporting interval. Because some transactions that were aborted in the last reporting interval may have started in a previous reporting interval, `Abort count` can sometimes be greater than `Trans count`.

- **`Scans`.** This is the number of table scans using this TC as the transaction coordinator that were started during the last reporting interval. This does not include range scans (that is, ordered index scans).

- **`Range scans`.** This is the number of ordered index scans using this TC as the transaction coordinator that were started in the last reporting interval.

Local query handler statistics (`Operations`). There is 1 cluster event per local query handler block (that is, 1 per data node process). Operations are recorded in the LQH where the data they are operating on resides.

Note

A single transaction may operate on data stored in multiple LQH blocks.

The `Operations` statistic provides the number of local operations performed by this LQH block in the last reporting interval, and includes all types of read and write operations (insert, update, write, and delete operations). This also includes operations used to replicate writes—for example, in a 2-replica cluster, the write to the primary replica is recorded in the primary LQH, and the write to the backup will be recorded in the backup LQH. Unique key operations may result in multiple local operations; however, this does *not* include local operations generated as a result of a table scan or ordered index scan, which are not counted.

Process scheduler statistics. In addition to the statistics reported by the transaction coordinator and local query handler, each `ndbd` process has a scheduler which also provides useful metrics relating to the performance of a MySQL Cluster. This scheduler runs in an infinite loop; during each loop the scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.
2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

Process scheduler statistics include the following:

- **`Mean Loop Counter`.** This is the number of loops executed in the third step from the preceding list. This statistic increases in size as the utilization of the TCP/IP buffer improves. You can use this to monitor changes in performance as you add new data node processes.
- **`Mean send size` and `Mean receive size`.** These statistics enable you to gauge the efficiency of, respectively writes and reads between nodes. The values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum value is 64K.

To cause all cluster log statistics to be logged, you can use the following command in the `NDB` management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

Note

Setting the threshold for `STATISTICS` to 15 causes the cluster log to become very verbose, and to grow quite rapidly in size, in direct proportion to the number of cluster nodes and the amount of activity in the MySQL Cluster.

For more information about MySQL Cluster management client commands relating to logging and reporting, see Section 15.5.5.1, “MySQL Cluster Logging Management Commands”.

15.5.6 MySQL Cluster Log Messages

This section contains information about the messages written to the cluster log in response to different cluster log events. It provides additional, more specific information on `NDB` transporter errors.

15.5.6.1 MySQL Cluster: Messages in the Cluster Log

The following table lists the most common `NDB` cluster log messages. For information about the cluster log, log events, and event types, see Section 15.5.5, “Event Reports Generated in MySQL Cluster”. These log messages also correspond to log event types in the MGM API; see The `Ndb_logevent_type` Type, for related information of interest to Cluster API developers.

Log Message. `Node mgm_node_id: Node data_node_id Connected` **Description.** The data node having node ID *`node_id`* has connected to the management server (node *`mgm_node_id`*).	**Event Name.** `Connected` **Event Type.** `Connection` **Priority.** 8 **Severity.** `INFO`
Log Message. `Node mgm_node_id: Node data_node_id Disconnected` **Description.** The data node having node ID *`data_node_id`* has disconnected from the management server (node *`mgm_node_id`*).	**Event Name.** `Disconnected` **Event Type.** `Connection` **Priority.** 8 **Severity.** `ALERT`
Log Message. `Node data_node_id: Communication to Node api_node_id closed` **Description.** The API node or SQL node having node ID *`api_node_id`* is no longer communicating with data node *`data_node_id`*.	**Event Name.** `CommunicationClosed` **Event Type.** `Connection` **Priority.** 8 **Severity.** `INFO`

Log Message. Node *data_node_id*: Communication to Node *api_node_id* opened **Description.** The API node or SQL node having node ID *api_node_id* is now communicating with data node *data_node_id*.	**Event Name.** CommunicationOpened **Event Type.** Connection **Priority.** 8 **Severity.** INFO
Log Message. Node *mgm_node_id*: Node *api_node_id*: API *version* **Description.** The API node having node ID *api_node_id* has connected to management node *mgm_node_id* using NDB API version *version* (generally the same as the MySQL version number).	**Event Name.** ConnectedApiVersion **Event Type.** Connection **Priority.** 8 **Severity.** INFO
Log Message. Node *node_id*: Global checkpoint *gci* started **Description.** A global checkpoint with the ID *gci* has been started; node *node_id* is the master responsible for this global checkpoint.	**Event Name.** GlobalCheckpointStarted **Event Type.** Checkpoint **Priority.** 9 **Severity.** INFO
Log Message. Node *node_id*: Global checkpoint *gci* completed **Description.** The global checkpoint having the ID *gci* has been completed; node *node_id* was the master responsible for this global checkpoint.	**Event Name.** GlobalCheckpointCompleted **Event Type.** Checkpoint **Priority.** 10 **Severity.** INFO
Log Message. Node *node_id*: Local checkpoint *lcp* started. Keep GCI = *current_gci* oldest restorable GCI = *old_gci* **Description.** The local checkpoint having sequence ID *lcp* has been started on node *node_id*. The most recent GCI that can be used has the index *current_gci*, and the oldest GCI from which the cluster can be restored has the index *old_gci*.	**Event Name.** LocalCheckpointStarted **Event Type.** Checkpoint **Priority.** 7 **Severity.** INFO
Log Message. Node *node_id*: Local checkpoint *lcp* completed **Description.** The local checkpoint having sequence ID *lcp* on node *node_id* has been completed.	**Event Name.** LocalCheckpointCompleted **Event Type.** Checkpoint **Priority.** 8 **Severity.** INFO
Log Message. Node *node_id*: Local Checkpoint stopped in CALCULATED_KEEP_GCI **Description.** The node was unable to determine the most recent usable GCI.	**Event Name.** LCPStoppedInCalcKeepGci **Event Type.** Checkpoint **Priority.** 0

	Severity. ALERT
Log Message. Node *node_id*: Table ID = *table_id*, fragment *ID* = *fragment_id* has completed LCP on Node *node_id* maxGciStarted: *started_gci* maxGciCompleted: *completed_gci* **Description.** A table fragment has been checkpointed to disk on node *node_id*. The GCI in progress has the index *started_gci*, and the most recent GCI to have been completed has the index *completed_gci*.	**Event Name.** LCPFragmentCompleted **Event Type.** Checkpoint **Priority.** 11 **Severity.** INFO
Log Message. Node *node_id*: ACC Blocked *num_1* and TUP Blocked *num_2* times last second **Description.** Undo logging is blocked because the log buffer is close to overflowing.	**Event Name.** UndoLogBlocked **Event Type.** Checkpoint **Priority.** 7 **Severity.** INFO
Log Message. Node *node_id*: Start initiated *version* **Description.** Data node *node_id*, running NDB version *version*, is beginning its startup process.	**Event Name.** NDBStartStarted **Event Type.** StartUp **Priority.** 1 **Severity.** INFO
Log Message. Node *node_id*: Started *version* **Description.** Data node *node_id*, running NDB version *version*, has started successfully.	**Event Name.** NDBStartCompleted **Event Type.** StartUp **Priority.** 1 **Severity.** INFO
Log Message. Node *node_id*: STTORRY received after restart finished **Description.** The node has received a signal indicating that a cluster restart has completed.	**Event Name.** STTORRYRecieved **Event Type.** StartUp **Priority.** 15 **Severity.** INFO
Log Message. Node *node_id*: Start phase *phase* completed (*type*) **Description.** The node has completed start phase *phase* of a *type* start. For a listing of start phases, see Section 15.5.1, "Summary of MySQL Cluster Start Phases". (*type* is one of initial, system [566], node, initial node, or <Unknown>.)	**Event Name.** StartPhaseCompleted **Event Type.** StartUp **Priority.** 4 **Severity.** INFO
Log Message. Node *node_id*: CM_REGCONF president = *president_id*, own Node = *own_id*, our dynamic id = *dynamic_id* **Description.** Node *president_id* has been selected as "president". *own_id* and *dynamic_id* should always be the same as the ID (*node_id*) of the reporting node.	**Event Name.** CM_REGCONF **Event Type.** StartUp **Priority.** 3 **Severity.** INFO

Log Message. `Node node_id: CM_REGREF from Node president_id to our Node node_id. Cause = cause` **Description.** The reporting node (ID `node_id`) was unable to accept node `president_id` as president. The `cause` of the problem is given as one of `Busy`, `Election with wait = false`, `Not president`, `Election without selecting new candidate`, or `No such cause`.	**Event Name.** `CM_REGREF` **Event Type.** `StartUp` **Priority.** 8 **Severity.** `INFO`
Log Message. `Node node_id: We are Node own_id with dynamic ID dynamic_id, our left neighbour is Node id_1, our right is Node id_2` **Description.** The node has discovered its neighboring nodes in the cluster (node `id_1` and node `id_2`). `node_id`, `own_id`, and `dynamic_id` should always be the same; if they are not, this indicates a serious misconfiguration of the cluster nodes.	**Event Name.** `FIND_NEIGHBOURS` **Event Type.** `StartUp` **Priority.** 8 **Severity.** `INFO`
Log Message. `Node node_id: type shutdown initiated` **Description.** The node has received a shutdown signal. The `type` of shutdown is either `Cluster` or `Node`.	**Event Name.** `NDBStopStarted` **Event Type.** `StartUp` **Priority.** 1 **Severity.** `INFO`
Log Message. `Node node_id: Node shutdown completed [, action] [Initiated by signal signal.]` **Description.** The node has been shut down. This report may include an `action`, which if present is one of `restarting`, `no start`, or `initial`. The report may also include a reference to an `NDB` Protocol `signal`; for possible signals, refer to Operations and Signals.	**Event Name.** `NDBStopCompleted` **Event Type.** `StartUp` **Priority.** 1 **Severity.** `INFO`
Log Message. `Node node_id: Forced node shutdown completed [, action]. [Occured during startphase start_phase.] [ Initiated by signal.] [Caused by error error_code: 'error_message(error_classification). error_status'. [(extra info extra_code)]]` **Description.** The node has been forcibly shut down. The `action` (one of `restarting`, `no start`, or `initial`) subsequently being taken, if any, is also reported. If the shutdown occurred while the node was starting, the report includes the `start_phase` during which the node failed. If this was a result of a `signal` sent to the node, this information is also provided (see Operations and Signals, for more information). If the error causing the failure is known, this is also included; for more information about `NDB` error messages and classifications, see MySQL Cluster API Errors.	**Event Name.** `NDBStopForced` **Event Type.** `StartUp` **Priority.** 1 **Severity.** `ALERT`
Log Message. `Node node_id: Node shutdown aborted`	**Event Name.** `NDBStopAborted`

Description. The node shutdown process was aborted by the user.	**Event Type.** StartUp **Priority.** 1 **Severity.** INFO
Log Message. Node *node_id*: StartLog: [GCI Keep: *keep_pos* LastCompleted: *last_pos* NewestRestorable: *restore_pos*] **Description.** This reports global checkpoints referenced during a node start. The redo log prior to *keep_pos* is dropped. *last_pos* is the last global checkpoint in which data node the participated; *restore_pos* is the global checkpoint which is actually used to restore all data nodes.	**Event Name.** StartREDOLog **Event Type.** StartUp **Priority.** 4 **Severity.** INFO
Log Message. *startup_message* [*Listed separately; see below.*] **Description.** There are a number of possible startup messages that can be logged under different circumstances.	**Event Name.** StartReport **Event Type.** StartUp **Priority.** 4 **Severity.** INFO
Log Message. Node *node_id*: Node restart completed copy of dictionary information **Description.** Copying of data dictionary information to the restarted node has been completed.	**Event Name.** NR_CopyDict **Event Type.** NodeRestart **Priority.** 8 **Severity.** INFO
Log Message. Node *node_id*: Node restart completed copy of distribution information **Description.** Copying of data distribution information to the restarted node has been completed.	**Event Name.** NR_CopyDistr **Event Type.** NodeRestart **Priority.** 8 **Severity.** INFO
Log Message. Node *node_id*: Node restart starting to copy the fragments to Node *node_id* **Description.** Copy of fragments to starting data node *node_id* has begun	**Event Name.** NR_CopyFragsStarted **Event Type.** NodeRestart **Priority.** 8 **Severity.** INFO
Log Message. Node *node_id*: Table ID = *table_id*, fragment ID = *fragment_id* have been copied to Node *node_id* **Description.** Fragment *fragment_id* from table *table_id* has been copied to data node *node_id*	**Event Name.** NR_CopyFragDone **Event Type.** NodeRestart **Priority.** 10 **Severity.** INFO
Log Message. Node *node_id*: Node restart completed copying the fragments to Node *node_id*	**Event Name.** NR_CopyFragsCompleted **Event Type.** NodeRestart

Description. Copying of all table fragments to restarting data node *node_id* has been completed	**Priority.** 8 **Severity.** INFO
Log Message. Any of the following: 1. Node *node_id*: Node *node1_id* completed failure of Node *node2_id* 2. All nodes completed failure of Node *node_id* 3. Node failure of *node_idblock* completed **Description.** One of the following (each corresponding to the same-numbered message listed above): 1. Data node *node1_id* has detected the failure of data node *node2_id* 2. All (remaining) data nodes have detected the failure of data node *node_id* 3. The failure of data node *node_id* has been detected in the *block*NDB kernel block, where block is 1 of DBTC, DBDICT, DBDIH, or DBLQH; for more information, see NDB Kernel Blocks	**Event Name.** NodeFailCompleted **Event Type.** NodeRestart **Priority.** 8 **Severity.** ALERT
Log Message. Node *mgm_node_id*: Node *data_node_id* has failed. The Node state at failure was *state_code* **Description.** A data node has failed. Its state at the time of failure is described by an arbitration state code *state_code*: possible state code values can be found in the file include/kernel/signaldata/ArbitSignalData.hpp.	**Event Name.** NODE_FAILREP **Event Type.** NodeRestart **Priority.** 8 **Severity.** ALERT
Log Message. President restarts arbitration thread [state=*state_code*] or Prepare arbitrator node *node_id* [ticket=*ticket_id*] or Receive arbitrator node *node_id* [ticket=*ticket_id*] or Started arbitrator node *node_id* [ticket=*ticket_id*] or Lost arbitrator node *node_id* - process failure [state=*state_code*] or Lost arbitrator node *node_id* - process exit [state=*state_code*] or Lost arbitrator node *node_id* - *error_message* [state=*state_code*] **Description.** This is a report on the current state and progress of arbitration in the cluster. *node_id* is the node ID of the management node or SQL node selected as the arbitrator. *state_code* is an arbitration state code, as found in include/kernel/signaldata/ArbitSignalData.hpp. When an error has occurred, an *error_message*, also defined in ArbitSignalData.hpp, is provided. *ticket_id* is a unique identifier handed out by the arbitrator when it is selected to all the nodes that participated in its selection; this is	**Event Name.** ArbitState **Event Type.** NodeRestart **Priority.** 6 **Severity.** INFO

used to ensure that each node requesting arbitration was one of the nodes that took part in the selection process.	
Log Message. `Arbitration check lost - less than 1/2 nodes left` or `Arbitration check won - all node groups and more than 1/2 nodes left` or `Arbitration check won - node group majority` or `Arbitration check lost - missing node group` or `Network partitioning - arbitration required` or `Arbitration won - positive reply from node` *`node_id`* or `Arbitration lost - negative reply from node` *`node_id`* or `Network partitioning - no arbitrator available` or `Network partitioning - no arbitrator configured` or `Arbitration failure - error_message` [`state=`*`state_code`*] **Description.** This message reports on the result of arbitration. In the event of arbitration failure, an *`error_message`* and an arbitration *`state_code`* are provided; definitions for both of these are found in `include/kernel/signaldata/ArbitSignalData.hpp`.	**Event Name.** `ArbitResult` **Event Type.** `NodeRestart` **Priority.** 2 **Severity.** `ALERT`
Log Message. `Node` *`node_id`*`: GCP Take over started` **Description.** This node is attempting to assume responsibility for the next global checkpoint (that is, it is becoming the master node)	**Event Name.** `GCP_TakeoverStarted` **Event Type.** `NodeRestart` **Priority.** 7 **Severity.** `INFO`
Log Message. `Node` *`node_id`*`: GCP Take over completed` **Description.** This node has become the master, and has assumed responsibility for the next global checkpoint	**Event Name.** `GCP_TakeoverCompleted` **Event Type.** `NodeRestart` **Priority.** 7 **Severity.** `INFO`
Log Message. `Node` *`node_id`*`: LCP Take over started` **Description.** This node is attempting to assume responsibility for the next set of local checkpoints (that is, it is becoming the master node)	**Event Name.** `LCP_TakeoverStarted` **Event Type.** `NodeRestart` **Priority.** 7 **Severity.** `INFO`
Log Message. `Node` *`node_id`*`: LCP Take over completed` **Description.** This node has become the master, and has assumed responsibility for the next set of local checkpoints	**Event Name.** `LCP_TakeoverCompleted` **Event Type.** `NodeRestart` **Priority.** 7 **Severity.** `INFO`

Log Message. Node *node_id*: Trans. Count = *transactions*, Commit Count = *commits*, Read Count = *reads*, Simple Read Count = *simple_reads*, Write Count = *writes*, AttrInfo Count = *AttrInfo_objects*, Concurrent Operations = *concurrent_operations*, Abort Count = *aborts*, Scans = *scans*, Range scans = *range_scans* **Description.** This report of transaction activity is given approximately once every 10 seconds	**Event Name.** TransReportCounters **Event Type.** Statistic **Priority.** 8 **Severity.** INFO
Log Message. Node *node_id*: Operations=*operations* **Description.** Number of operations performed by this node, provided approximately once every 10 seconds	**Event Name.** OperationReportCounters **Event Type.** Statistic **Priority.** 8 **Severity.** INFO
Log Message. Node *node_id*: Table with ID = *table_id* created **Description.** A table having the table ID shown has been created	**Event Name.** TableCreated **Event Type.** Statistic **Priority.** 7 **Severity.** INFO
Log Message. Node *node_id*: Mean loop Counter in doJob last 8192 times = *count* **Description.**	**Event Name.** JobStatistic **Event Type.** Statistic **Priority.** 9 **Severity.** INFO
Log Message. Mean send size to Node = *node_id* last 4096 sends = *bytes* bytes **Description.** This node is sending an average of *bytes* bytes per send to node *node_id*	**Event Name.** SendBytesStatistic **Event Type.** Statistic **Priority.** 9 **Severity.** INFO
Log Message. Mean receive size to Node = *node_id* last 4096 sends = *bytes* bytes **Description.** This node is receiving an average of *bytes* of data each time it receives data from node *node_id*	**Event Name.** ReceiveBytesStatistic **Event Type.** Statistic **Priority.** 9 **Severity.** INFO
Log Message. Node *node_id*: Data usage is *data_memory_percentage*% (*data_pages_used* 32K pages of total *data_pages_total*) / Node *node_id*: Index usage is *index_memory_percentage*% (*index_pages_used* 8K pages of total *index_pages_total*)	**Event Name.** MemoryUsage **Event Type.** Statistic **Priority.** 5

Description. This report is generated when a `DUMP 1000` command is issued in the cluster management client; for more information, see `DUMP 1000`, in MySQL Cluster Internals	**Severity.** `INFO`
Log Message. `Node node1_id: Transporter to node node2_id reported error error_code: error_message` **Description.** A transporter error occurred while communicating with node *`node2_id`*; for a listing of transporter error codes and messages, see NDB Transporter Errors, in MySQL Cluster Internals	**Event Name.** `TransporterError` **Event Type.** `Error` **Priority.** 2 **Severity.** `ERROR`
Log Message. `Node node1_id: Transporter to node node2_id reported error error_code: error_message` **Description.** A warning of a potential transporter problem while communicating with node *`node2_id`*; for a listing of transporter error codes and messages, see NDB Transporter Errors, for more information	**Event Name.** `TransporterWarning` **Event Type.** `Error` **Priority.** 8 **Severity.** `WARNING`
Log Message. `Node node1_id: Node node2_id missed heartbeat heartbeat_id` **Description.** This node missed a heartbeat from node *`node2_id`*	**Event Name.** `MissedHeartbeat` **Event Type.** `Error` **Priority.** 8 **Severity.** `WARNING`
Log Message. `Node node1_id: Node node2_id declared dead due to missed heartbeat` **Description.** This node has missed at least 3 heartbeats from node *`node2_id`*, and so has declared that node “dead”	**Event Name.** `DeadDueToHeartbeat` **Event Type.** `Error` **Priority.** 8 **Severity.** `ALERT`
Log Message. `Node node1_id: Node Sent Heartbeat to node = node2_id` **Description.** This node has sent a heartbeat to node *`node2_id`*	**Event Name.** `SentHeartbeat` **Event Type.** `Info` **Priority.** 12 **Severity.** `INFO`
Log Message. `Node node_id: Event buffer status: used=bytes_used (percent_used%) alloc=bytes_allocated (percent_available%) max=bytes_available apply_gci=latest_restorable_GCI latest_gci=latest_GCI` **Description.** This report is seen during heavy event buffer usage, for example, when many updates are being applied in a relatively short period of time; the report shows the number of bytes and the percentage of event buffer memory used, the bytes allocated and percentage still available, and the latest and latest restorable global checkpoints	**Event Name.** `EventBufferStatus` **Event Type.** `Info` **Priority.** 7 **Severity.** `INFO`

Log Message. `Node node_id: Entering single user mode`, `Node node_id: Entered single user mode` `Node API_node_id has exclusive access`, `Node node_id: Entering single user mode` **Description.** These reports are written to the cluster log when entering and exiting single user mode; *`API_node_id`* is the node ID of the API or SQL having exclusive access to the cluster (fro mroe information, see Section 15.5.7, "MySQL Cluster Single User Mode"); the message `Unknown single user report API_node_id` indicates an error has taken place and should never be seen in normal operation	**Event Name.** `SingleUser` **Event Type.** `Info` **Priority.** 7 **Severity.** `INFO`
Log Message. `Node node_id: Backup backup_id started from node mgm_node_id` **Description.** A backup has been started using the management node having *`mgm_node_id`*; this message is also displayed in the cluster management client when the `START BACKUP` command is issued; for more information, see Section 15.5.3.2, "Using The MySQL Cluster Management Client to Create a Backup"	**Event Name.** `BackupStarted` **Event Type.** `Backup` **Priority.** 7 **Severity.** `INFO`
Log Message. `Node node_id: Backup backup_id started from node mgm_node_id completed. StartGCP: start_gcp StopGCP: stop_gcp #Records: records #LogRecords: log_records Data: data_bytes bytes Log: log_bytes bytes` **Description.** The backup having the ID *`backup_id`* has been completed; for more information, see Section 15.5.3.2, "Using The MySQL Cluster Management Client to Create a Backup"	**Event Name.** `BackupCompleted` **Event Type.** `Backup` **Priority.** 7 **Severity.** `INFO`
Log Message. `Node node_id: Backup request from mgm_node_id failed to start. Error: error_code` **Description.** The backup failed to start; for error codes, see MGM API Errors	**Event Name.** `BackupFailedToStart` **Event Type.** `Backup` **Priority.** 7 **Severity.** `ALERT`
Log Message. `Node node_id: Backup backup_id started from mgm_node_id has been aborted. Error: error_code` **Description.** The backup was terminated after starting, possibly due to user intervention	**Event Name.** `BackupAborted` **Event Type.** `Backup` **Priority.** 7 **Severity.** `ALERT`

15.5.6.2 MySQL Cluster: `NDB` Transporter Errors

This section lists error codes, names, and messages that are written to the cluster log in the event of transporter errors.

Error Code	Error Name	Error Text
0x00	**TE_NO_ERROR**	No error
0x01	**TE_ERROR_CLOSING_SOCKET**	Error found during closing of socket
0x02	**TE_ERROR_IN_SELECT_BEFORE_ACCEPT**	Error found before accept. The transporter will retry
0x03	**TE_INVALID_MESSAGE_LENGTH**	Error found in message (invalid message length)
0x04	**TE_INVALID_CHECKSUM**	Error found in message (checksum)
0x05	**TE_COULD_NOT_CREATE_SOCKET**	Error found while creating socket(can't create socket)
0x06	**TE_COULD_NOT_BIND_SOCKET**	Error found while binding server socket
0x07	**TE_LISTEN_FAILED**	Error found while listening to server socket
0x08	**TE_ACCEPT_RETURN_ERROR**	Error found during accept(accept return error)
0x0b	**TE_SHM_DISCONNECT**	The remote node has disconnected
0x0c	**TE_SHM_IPC_STAT**	Unable to check shm segment
0x0d	**TE_SHM_UNABLE_TO_CREATE_SEGMENT**	Unable to create shm segment
0x0e	**TE_SHM_UNABLE_TO_ATTACH_SEGMENT**	Unable to attach shm segment
0x0f	**TE_SHM_UNABLE_TO_REMOVE_SEGMENT**	Unable to remove shm segment
0x10	**TE_TOO_SMALL_SIGID**	Sig ID too small
0x11	**TE_TOO_LARGE_SIGID**	Sig ID too large
0x12	**TE_WAIT_STACK_FULL**	Wait stack was full
0x13	**TE_RECEIVE_BUFFER_FULL**	Receive buffer was full
0x14	**TE_SIGNAL_LOST_SEND_BUFFER_FULL**	Send buffer was full,and trying to force send fails
0x15	**TE_SIGNAL_LOST**	Send failed for unknown reason(signal lost)

Error Code	Error Name	Error Text
0x16	**TE_SEND_BUFFER_FULL**	The send buffer was full, but sleeping for a while solved
0x0017	**TE_SCI_LINK_ERROR**	There is no link from this node to the switch
0x18	**TE_SCI_UNABLE_TO_START_SEQUENCE**	Could not start a sequence, because system resources are exumed or no sequence has been created
0x19	**TE_SCI_UNABLE_TO_REMOVE_SEQUENCE**	Could not remove a sequence
0x1a	**TE_SCI_UNABLE_TO_CREATE_SEQUENCE**	Could not create a sequence, because system resources are exempted. Must reboot
0x1b	**TE_SCI_UNRECOVERABLE_DATA_TFX_ERROR**	Tried to send data on redundant link but failed
0x1c	**TE_SCI_CANNOT_INIT_LOCALSEGMENT**	Cannot initialize local segment
0x1d	**TE_SCI_CANNOT_MAP_REMOTESEGMENT**	Cannot map remote segment
0x1e	**TE_SCI_UNABLE_TO_UNMAP_SEGMENT**	Cannot free the resources used by this segment (step 1)
0x1f	**TE_SCI_UNABLE_TO_REMOVE_SEGMENT**	Cannot free the resources used by this segment (step 2)
0x20	**TE_SCI_UNABLE_TO_DISCONNECT_SEGMENT**	Cannot disconnect from a remote segment
0x21	**TE_SHM_IPC_PERMANENT**	Shm ipc Permanent error
0x22	**TE_SCI_UNABLE_TO_CLOSE_CHANNEL**	Unable to close the sci channel and the resources allocated

15.5.7 MySQL Cluster Single User Mode

Single user mode enables the database administrator to restrict access to the database system to a single API node, such as a MySQL server (SQL node) or an instance of `ndb_restore`. When entering single user mode, connections to all other API nodes are closed gracefully and all running transactions are aborted. No new transactions are permitted to start.

Once the cluster has entered single user mode, only the designated API node is granted access to the database.

You can use the `ALL STATUS` command to see when the cluster has entered single user mode.

Example:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the API node whose node ID is `5` becomes the cluster's only permitted user.

The node specified in the preceding command must be an API node; attempting to specify any other type of node will be rejected.

Note

When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command `EXIT SINGLE USER MODE` changes the state of the cluster's data nodes from single user mode to normal mode. API nodes—such as MySQL Servers—waiting for a connection (that is, waiting for the cluster to become ready and available), are again permitted to connect. The API node denoted as the single-user node continues to run (if still connected) during and after the state change.

Example:

```
ndb_mgm> EXIT SINGLE USER MODE
```

There are two recommended ways to handle a node failure when running in single user mode:

- Method 1:

  1. Finish all single user mode transactions
  2. Issue the `EXIT SINGLE USER MODE` command
  3. Restart the cluster's data nodes

- Method 2:

  Restart database nodes prior to entering single user mode.

15.5.8 Quick Reference: MySQL Cluster SQL Statements

This section discusses several SQL statements that can prove useful in managing and monitoring a MySQL server that is connected to a MySQL Cluster, and in some cases provide information about the cluster itself.

- `SHOW ENGINE NDB STATUS`, `SHOW ENGINE NDBCLUSTER STATUS`

  The output of this statement contains information about the server's connection to the cluster, creation and usage of MySQL Cluster objects, and binary logging for MySQL Cluster replication.

  See Section 12.4.5.9, “`SHOW ENGINE` Syntax”, for a usage example and more detailed information.

- `SHOW ENGINES [LIKE 'NDB%']`

This statement can be used to determine whether or not clustering support is enabled in the MySQL server, and if so, whether it is active.

See Section 12.4.5.10, “SHOW ENGINES Syntax”, for more detailed information.

- `SHOW VARIABLES LIKE 'NDB%'`

  This statement provides a list of most server system variables relating to the `NDB` storage engine, and their values, as shown here:

```
mysql> SHOW VARIABLES LIKE 'NDB%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| ndb_autoincrement_prefetch_sz     | 32    |
| ndb_cache_check_time              | 0     |
| ndb_extra_logging                 | 0     |
| ndb_force_send                    | ON    |
| ndb_index_stat_cache_entries      | 32    |
| ndb_index_stat_enable             | OFF   |
| ndb_index_stat_update_freq        | 20    |
| ndb_report_thresh_binlog_epoch_slip | 3     |
| ndb_report_thresh_binlog_mem_usage  | 10    |
| ndb_use_copying_alter_table       | OFF   |
| ndb_use_exact_count               | ON    |
| ndb_use_transactions              | ON    |
+-----------------------------------+-------+
```

  See Section 5.1.3, “Server System Variables”, for more information.

- `SHOW STATUS LIKE 'NDB%'`

  This statement shows at a glance whether or not the MySQL server is acting as a cluster SQL node, and if so, it provides the MySQL server's cluster node ID, the host name and port for the cluster management server to which it is connected, and the number of data nodes in the cluster, as shown here:

```
mysql> SHOW STATUS LIKE 'NDB%';
+--------------------------+---------------+
| Variable_name            | Value         |
+--------------------------+---------------+
| Ndb_cluster_node_id      | 10            |
| Ndb_config_from_host     | 192.168.0.103 |
| Ndb_config_from_port     | 1186          |
| Ndb_number_of_data_nodes | 4             |
+--------------------------+---------------+
```

  If the MySQL server was built with clustering support, but it is not connected to a cluster, all rows in the output of this statement contain a zero or an empty string:

```
mysql> SHOW STATUS LIKE 'NDB%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Ndb_cluster_node_id      | 0     |
| Ndb_config_from_host     |       |
| Ndb_config_from_port     | 0     |
| Ndb_number_of_data_nodes | 0     |
+--------------------------+-------+
```

See also Section 12.4.5.22, “SHOW STATUS Syntax”.

15.5.9 MySQL Cluster Security Issues

This section discusses security considerations to take into account when setting up and running MySQL Cluster.

Topics to be covered in this chapter include the following:

- MySQL Cluster and network security issues
- Configuration issues relating to running MySQL Cluster securely
- MySQL Cluster and the MySQL privilege system
- MySQL standard security procedures as applicable to MySQL Cluster

15.5.9.1 MySQL Cluster Security and Networking Issues

In this section, we discuss basic network security issues as they relate to MySQL Cluster. It is extremely important to remember that MySQL Cluster “out of the box” is not secure; you or your network administrator must take the proper steps to ensure that your cluster cannot be compromised over the network.

Cluster communication protocols are inherently insecure, and no encryption or similar security measures are used in communications between nodes in the cluster. Because network speed and latency have a direct impact on the cluster's efficiency, it is also not advisable to employ SSL or other encryption to network connections between nodes, as such schemes will effectively slow communications.

It is also true that no authentication is used for controlling API node access to a MySQL Cluster. As with encryption, the overhead of imposing authentication requirements would have an adverse impact on Cluster performance.

In addition, there is no checking of the source IP address for either of the following when accessing the cluster:

- SQL or API nodes using “free slots” created by empty `[mysqld]` or `[api]` sections in the `config.ini` file

  This means that, if there are any empty `[mysqld]` or `[api]` sections in the `config.ini` file, then any API nodes (including SQL nodes) that know the management server's host name (or IP address) and port can connect to the cluster and access its data without restriction. (See Section 15.5.9.2, “MySQL Cluster and MySQL Privileges”, for more information about this and related issues.)

Note

You can exercise some control over SQL and API node access to the cluster by specifying a `HostName` parameter for all `[mysqld]` and `[api]` sections in the `config.ini` file. However, this also means that, should you wish to connect an API node to the cluster from a previously unused host, you need to add an `[api]` section containing its host name to the `config.ini` file.

More information is available elsewhere in this chapter [1280] about the `HostName` parameter. Also see Section 15.3.1, “Quick Test Setup of MySQL Cluster”, for configuration examples using `HostName` with API nodes.

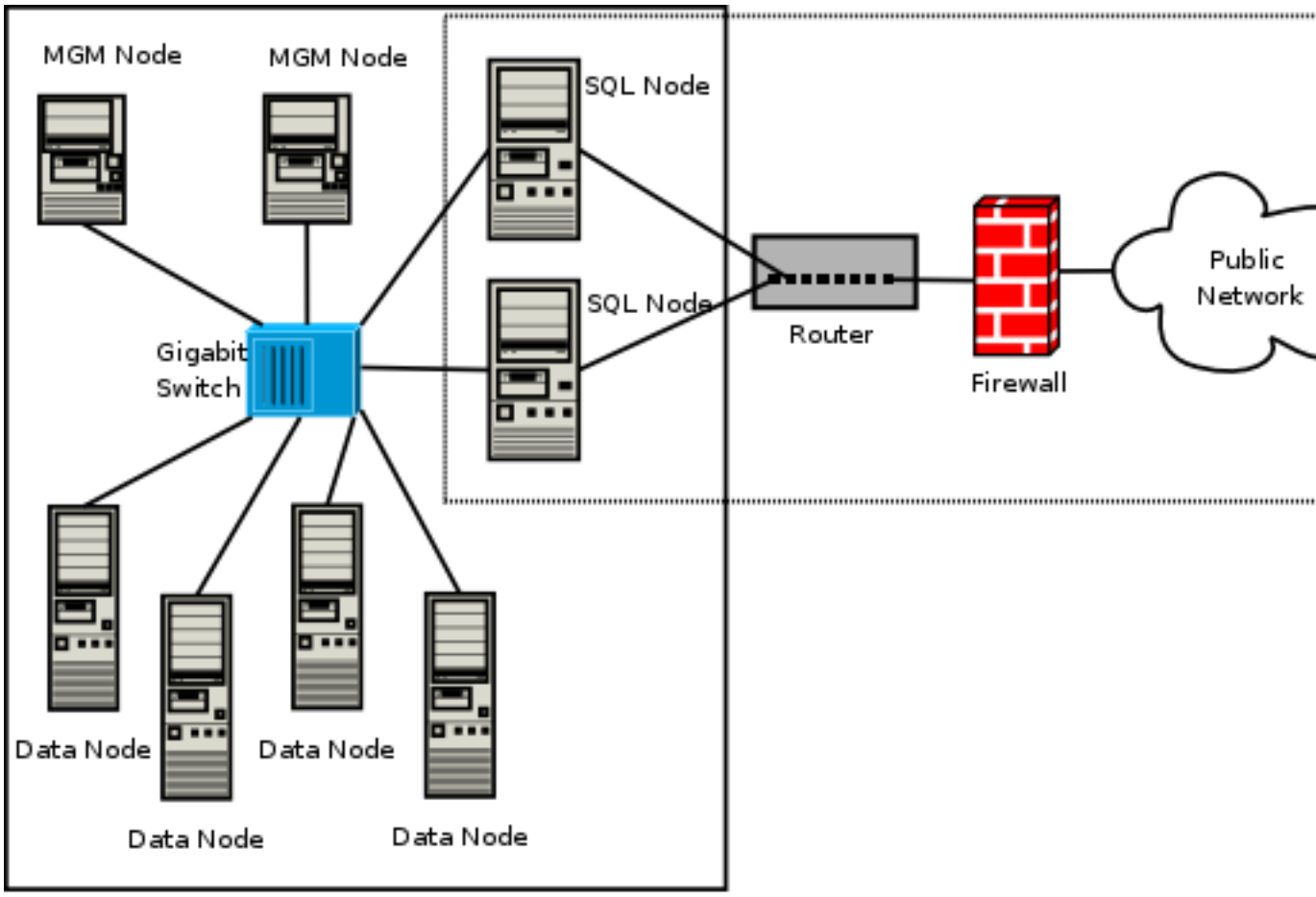
- Any `ndb_mgm` client

  This means that any cluster management client that is given the management server's host name (or IP address) and port (if not the standard port) can connect to the cluster and execute any management client command. This includes commands such as `ALL STOP` and `SHUTDOWN`.

For these reasons, it is necessary to protect the cluster on the network level. The safest network configuration for Cluster is one which isolates connections between Cluster nodes from any other network communications. This can be accomplished by any of the following methods:

1. Keeping Cluster nodes on a network that is physically separate from any public networks. This option is the most dependable; however, it is the most expensive to implement.

   We show an example of a MySQL Cluster setup using such a physically segregated network here:



   This setup has two networks, one private (solid box) for the Cluster management servers and data nodes, and one public (dotted box) where the SQL nodes reside. (We show the management and data nodes connected using a gigabit switch since this provides the best performance.) Both networks are protected from the outside by a hardware firewall, sometimes also known as a *network-based firewall*.

   This network setup is safest because no packets can reach the cluster's management or data nodes from outside the network—and none of the cluster's internal communications can reach the outside—without going through the SQL nodes, as long as the SQL nodes do not permit any packets to be forwarded. This means, of course, that all SQL nodes must be secured against hacking attempts.

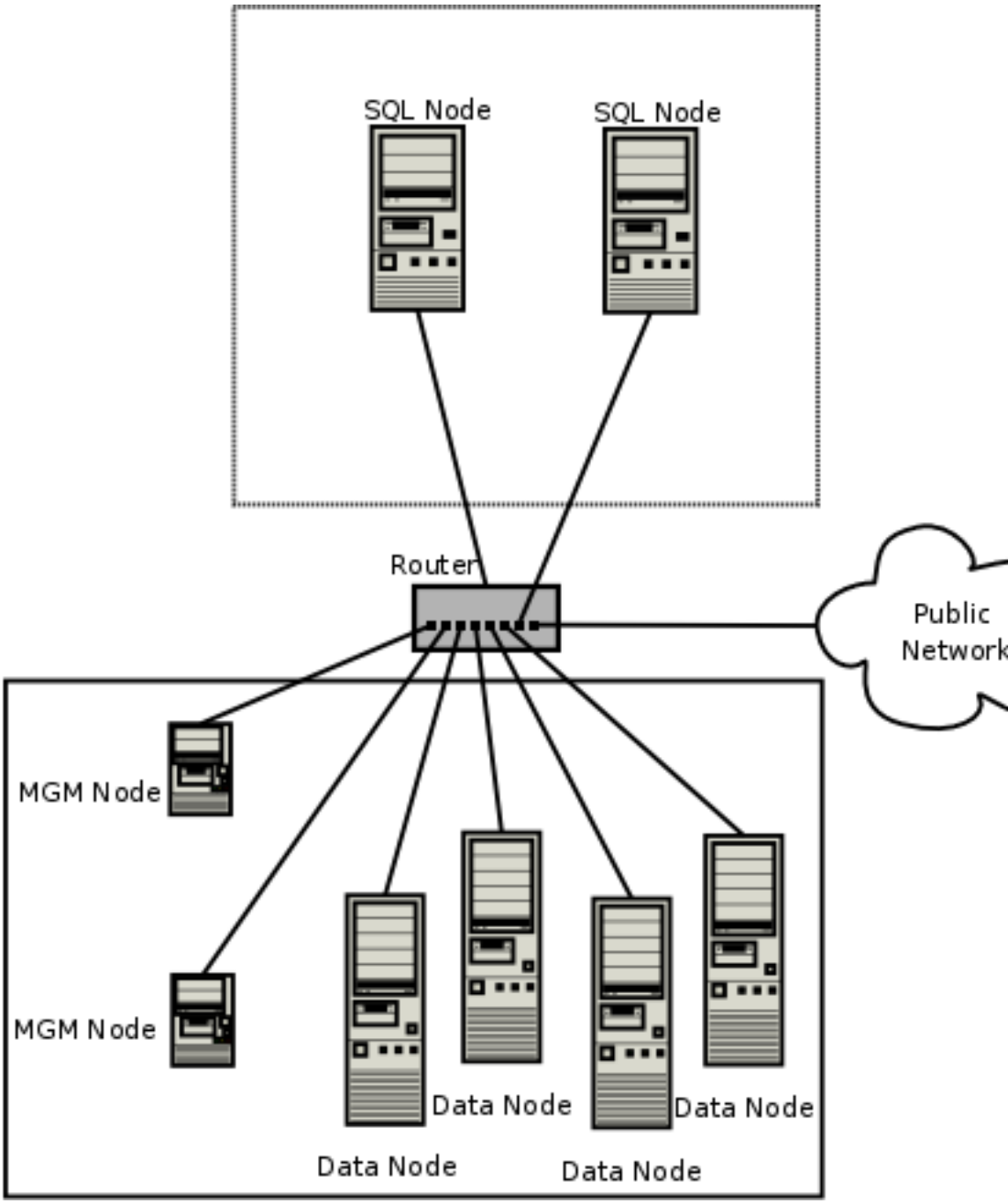
Important

With regard to potential security vulnerabilities, an SQL node is no different from any other MySQL server. See Section 5.4.3, “Making MySQL Secure Against Attackers”, for a description of techniques you can use to secure MySQL servers.

2. Using one or more software firewalls (also known as *host-based firewalls*) to control which packets pass through to the cluster from portions of the network that do not require access to it. In this type of setup, a software firewall must be installed on every host in the cluster which might otherwise be accessible from outside the local network.

   The host-based option is the least expensive to implement, but relies purely on software to provide protection and so is the most difficult to keep secure.

   This type of network setup for MySQL Cluster is illustrated here:



Using this type of network setup means that there are two zones of MySQL Cluster hosts. Each cluster host must be able to communicate with all of the other machines in the cluster, but only those hosting SQL nodes (dotted box) can be permitted to have any contact with the outside, while those in the zone containing the data nodes and management nodes (solid box) must be isolated from any machines that are not part of the cluster. Applications using the cluster and user of those applications must *not* be permitted to have direct access to the management and data node hosts.

To accomplish this, you must set up software firewalls that limit the traffic to the type or types shown in the following table, according to the type of node that is running on each cluster host computer:

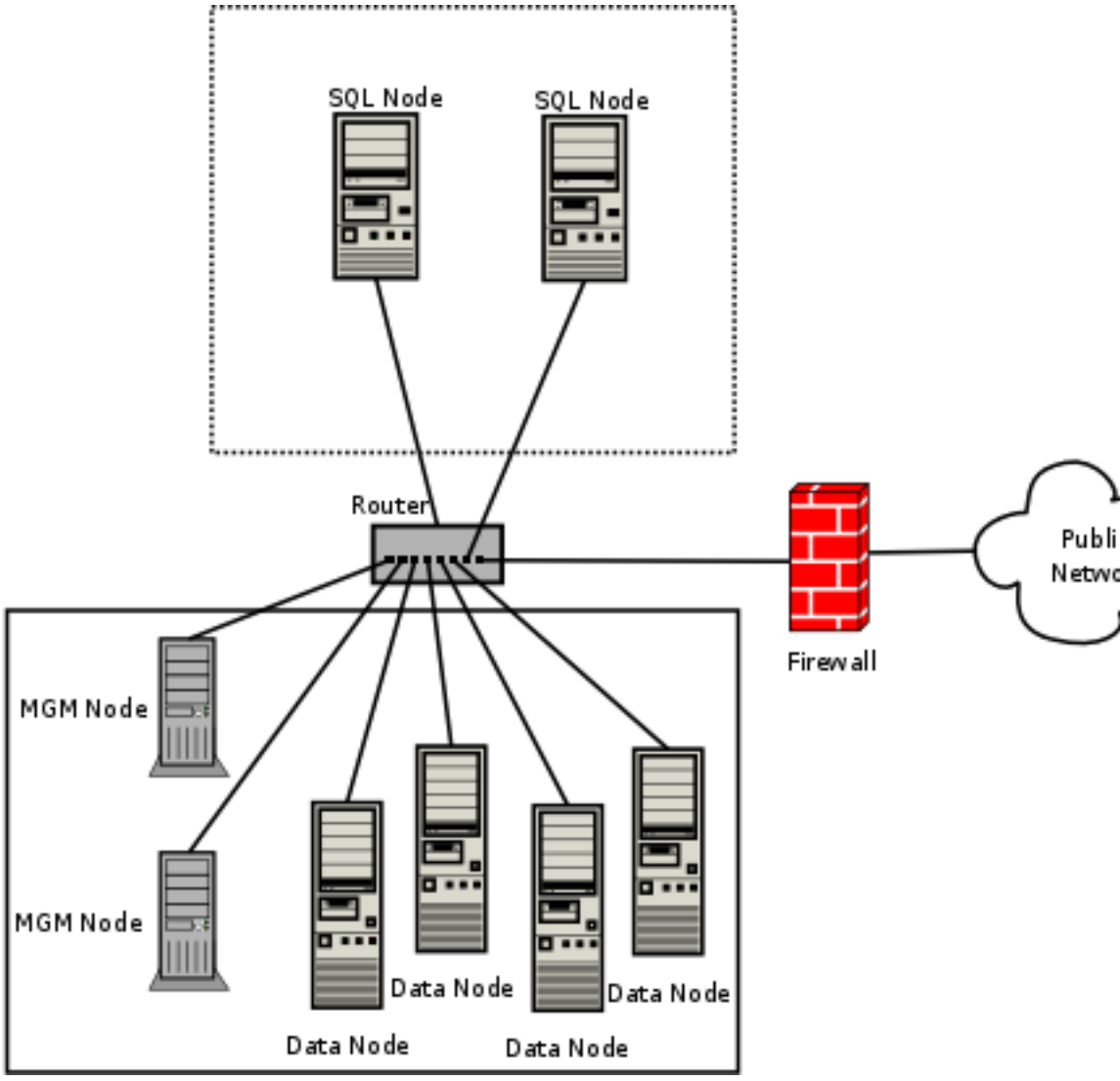
Type of Node to be Accessed	Traffic to Permit
SQL or API node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from within the network in which the cluster resides and is on the port that your application is using.
Data node or Management node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from the IP address of an SQL or API node.

Any traffic other than that shown in the table for a given node type should be denied.

The specifics of configuring a firewall vary from firewall application to firewall application, and are beyond the scope of this Manual. `iptables` is a very common and reliable firewall application, which is often used with `APF` as a front end to make configuration easier. You can (and should) consult the documentation for the software firewall that you employ, should you choose to implement a MySQL Cluster network setup of this type, or of a “mixed” type as discussed under the next item.

3. It is also possible to employ a combination of the first two methods, using both hardware and software to secure the cluster—that is, using both network-based and host-based firewalls. This is between the first two schemes in terms of both security level and cost. This type of network setup keeps the cluster behind the hardware firewall, but permits incoming packets to travel beyond the router connecting all cluster hosts to reach the SQL nodes.

   One possible network deployment of a MySQL Cluster using hardware and software firewalls in combination is shown here:



In this case, you can set the rules in the hardware firewall to deny any external traffic except to SQL nodes and API nodes, and then permit traffic to them only on the ports required by your application.

Whatever network configuration you use, remember that your objective from the viewpoint of keeping the cluster secure remains the same—to prevent any unessential traffic from reaching the cluster while ensuring the most efficient communication between the nodes in the cluster.

Because MySQL Cluster requires large numbers of ports to be open for communications between nodes, the recommended option is to use a segregated network. This represents the simplest way to prevent unwanted traffic from reaching the cluster.

Note

If you wish to administer a MySQL Cluster remotely (that is, from outside the local network), the recommended way to do this is to use `ssh` or another secure login shell to access an SQL node host. From this host, you can then run the management client to access the management server safely, from within the Cluster's own local network.

> Even though it is possible to do so in theory, it is *not* recommended to use `ndb_mgm` to manage a Cluster directly from outside the local network on which the Cluster is running. Since neither authentication nor encryption takes place between the management client and the management server, this represents an extremely insecure means of managing the cluster, and is almost certain to be compromised sooner or later.

15.5.9.2 MySQL Cluster and MySQL Privileges

In this section, we discuss how the MySQL privilege system works in relation to MySQL Cluster and the implications of this for keeping a MySQL Cluster secure.

Standard MySQL privileges apply to MySQL Cluster tables. This includes all MySQL privilege types (`SELECT` [492] privilege, `UPDATE` [493] privilege, `DELETE` [491] privilege, and so on) granted on the database, table, and column level. As with any other MySQL Server, user and privilege information is stored in the `mysql` system database. The SQL statements used to grant and revoke privileges on `NDB` tables, databases containing such tables, and columns within such tables are identical in all respects with the `GRANT` and `REVOKE` statements used in connection with database objects involving any (other) MySQL storage engine.

It is important to keep in mind that the MySQL grant tables use the `MyISAM` storage engine. Because of this, those tables are not duplicated or shared among MySQL servers acting as SQL nodes in a MySQL Cluster. By way of example, suppose that two SQL nodes **A** and **B** are connected to the same MySQL Cluster, which has an `NDB` table named `mytable` in a database named `mydb`, and that you execute an SQL statement on server **A** that creates a new user `jon@localhost` and grants this user the `SELECT` [492] privilege on that table:

```
mysql> GRANT SELECT ON mydb.mytable
    ->   TO jon@localhost IDENTIFIED BY 'mypass';
```

This user is *not* created on server **B**. For this to take place, the statement must also be run on server **B**. Similarly, statements run on server **A** and affecting the privileges of existing users on server **A** do not affect users on server **B** unless those statements are actually run on server **B** as well.

In other words, *changes in users and their privileges do not automatically propagate between SQL nodes*. Synchronization of privileges between SQL nodes must be done either manually or by scripting an application that periodically synchronizes the privilege tables on all SQL nodes in the cluster.

Conversely, because there is no way in MySQL to deny privileges (privileges can either be revoked or not granted in the first place, but not denied as such), there is no special protection for `NDB` tables on one SQL node from users that have privileges on another SQL node. The most far-reaching example of this is the MySQL `root` account, which can perform any action on any database object. In combination with empty `[mysqld]` or `[api]` sections of the `config.ini` file, this account can be especially dangerous. To understand why, consider the following scenario:

- The `config.ini` file contains at least one empty `[mysqld]` or `[api]` section. This means that the Cluster management server performs no checking of the host from which a MySQL Server (or other API node) accesses the MySQL Cluster.
- There is no firewall, or the firewall fails to protect against access to the Cluster from hosts external to the network.
- The host name or IP address of the Cluster's management server is known or can be determined from outside the network.

If these conditions are true, then anyone, anywhere can start a MySQL Server with `--ndbcluster --ndb-connectstring=*management_host*` and access the Cluster. Using the MySQL `root` account, this person can then perform the following actions:

- Execute a `SHOW DATABASES` statement to obtain a list of all databases that exist in the cluster
- Execute a `SHOW TABLES FROM *some_database*` statement to obtain a list of all `NDB` tables in a given database
- Run any legal MySQL statements on any of those tables, such as:
  - `SELECT * FROM *some_table*` to read all the data from any table
  - `DELETE FROM *some_table*` to delete all the data from a table
  - `DESCRIBE *some_table*` or `SHOW CREATE TABLE *some_table*` to determine the table schema
  - `UPDATE *some_table* SET *column1 = any_value1*` to fill a table column with “garbage” data; this could actually cause much greater damage than simply deleting all the data

    Even more insidious variations might include statements like these:

    ```
    UPDATE some_table SET an_int_column = an_int_column + 1
    ```

    or

    ```
    UPDATE some_table SET a_varchar_column = REVERSE(a_varchar_column)
    ```

    Such malicious statements are limited only by the imagination of the attacker.

  The only tables that would be safe from this sort of mayhem would be those tables that were created using storage engines other than `NDB`, and so not visible to a “rogue” SQL node.

Note

A user who can log in as `root` can also access the `INFORMATION_SCHEMA` database and its tables, and so obtain information about databases, tables, stored routines, scheduled events, and any other database objects for which metadata is stored in `INFORMATION_SCHEMA`.

  It is also a very good idea to use different passwords for the `root` accounts on different cluster SQL nodes.

In sum, you cannot have a safe MySQL Cluster if it is directly accessible from outside your local network.

Important

Never leave the MySQL root account password empty. This is just as true when running MySQL as a MySQL Cluster SQL node as it is when running it as a standalone (non-Cluster) MySQL Server, and should be done as part of the MySQL installation process before configuring the MySQL Server as an SQL node in a MySQL Cluster.

You should never convert the system tables in the `mysql` database to use the `NDB` storage engine. There are a number of reasons why you should not do this, but the most important reason is this: *Many of the SQL statements that affect `mysql` tables storing information about user privileges, stored routines, scheduled events, and other database objects cease to function if these tables are changed to use any*

storage engine other than `MyISAM`. This is a consequence of various MySQL Server internals which are not expected to change in the foreseeable future.

If you need to synchronize `mysql` system tables between SQL nodes, you can use standard MySQL replication to do so, or employ a script to copy table entries between the MySQL servers.

Summary. The two most important points to remember regarding the MySQL privilege system with regard to MySQL Cluster are:

1. Users and privileges established on one SQL node do not automatically exist or take effect on other SQL nodes in the cluster.

   Conversely, removing a user or privilege on one SQL node in the cluster does not remove the user or privilege from any other SQL nodes.

2. Once a MySQL user is granted privileges on an `NDB` table from one SQL node in a MySQL Cluster, that user can "see" any data in that table regardless of the SQL node from which the data originated.

15.5.9.3 MySQL Cluster and MySQL Security Procedures

In this section, we discuss MySQL standard security procedures as they apply to running MySQL Cluster.

In general, any standard procedure for running MySQL securely also applies to running a MySQL Server as part of a MySQL Cluster. First and foremost, you should always run a MySQL Server as the `mysql` system user; this is no different from running MySQL in a standard (non-Cluster) environment. The `mysql` system account should be uniquely and clearly defined. Fortunately, this is the default behavior for a new MySQL installation. You can verify that the `mysqld` process is running as the system user `mysql` by using the system command such as the one shown here:

```
shell> ps aux | grep mysql
root     10467  0.0  0.1   3616  1380 pts/3    S    11:53   0:00 \
  /bin/sh ./mysqld_safe --ndbcluster --ndb-connectstring=localhost:1186
mysql    10512  0.2  2.5  58528 26636 pts/3    Sl   11:53   0:00 \
  /usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql \
  --datadir=/usr/local/mysql/var --user=mysql --ndbcluster \
  --ndb-connectstring=localhost:1186 --pid-file=/usr/local/mysql/var/mothra.pid \
  --log-error=/usr/local/mysql/var/mothra.err
jon      10579  0.0  0.0   2736   688 pts/0    S+   11:54   0:00 grep mysql
```

If the `mysqld` process is running as any other user than `mysql`, you should immediately shut it down and restart it as the `mysql` user. If this user does not exist on the system, the `mysql` user account should be created, and this user should be part of the `mysql` user group; in this case, you should also make sure that the MySQL `DataDir` on this system is owned by the `mysql` user, and that the SQL node's `my.cnf` file includes `user=mysql` in the `[mysqld]` section. Alternatively, you can start the server with `--user=mysql` on the command line, but it is preferable to use the `my.cnf` option, since you might forget to use the command-line option and so have `mysqld` running as another user unintentionally. The `mysqld_safe` startup script forces MySQL to run as the `mysql` user.

Important

Never run `mysqld` as the system root user. Doing so means that potentially any file on the system can be read by MySQL, and thus—should MySQL be compromised—by an attacker.

As mentioned in the previous section (see Section 15.5.9.2, "MySQL Cluster and MySQL Privileges"), you should always set a root password for the MySQL Server as soon as you have it running. You should also delete the anonymous user account that is installed by default. You can accomplish these tasks using the following statements:

```
shell> mysql -u root

mysql> UPDATE mysql.user
    ->     SET Password=PASSWORD('secure_password')
    ->     WHERE User='root';

mysql> DELETE FROM mysql.user
    ->     WHERE User='';

mysql> FLUSH PRIVILEGES;
```

Be very careful when executing the `DELETE` statement not to omit the `WHERE` clause, or you risk deleting *all* MySQL users. Be sure to run the `FLUSH PRIVILEGES` statement as soon as you have modified the `mysql.user` table, so that the changes take immediate effect. Without `FLUSH PRIVILEGES`, the changes do not take effect until the next time that the server is restarted.

Note

Many of the MySQL Cluster utilities such as `ndb_show_tables`, `ndb_desc`, and `ndb_select_all` also work without authentication and can reveal table names, schemas, and data. By default these are installed on Unix-style systems with the permissions `wxr-xr-x` (755), which means they can be executed by any user that can access the `mysql/bin` directory.

See Section 15.4, “MySQL Cluster Programs”, for more information about these utilities.

15.6 MySQL 4.1 FAQ: MySQL Cluster

In the following section, we answer questions that are frequently asked about MySQL Cluster and the `NDBCLUSTER` storage engine.

Questions

- 16.6.1: [1376] Which versions of the MySQL software support Cluster? Do I have to compile from source?
- 16.6.2: [1376] What does “NDB” mean?
- 16.6.3: [1376] What is the difference between using MySQL Cluster *vs* using MySQL replication?
- 16.6.4: [1376] Do I need to do any special networking to run MySQL Cluster? How do computers in a cluster communicate?
- 16.6.5: [1377] How many computers do I need to run a MySQL Cluster, and why?
- 16.6.6: [1377] What do the different computers do in a MySQL Cluster?
- 16.6.7: [1377] When I run the `SHOW` command in the MySQL Cluster management client, I see a line of output that looks like this:

  ```
  id=2    @10.100.10.32  (Version: 4.1.25, Nodegroup: 0, Master)
  ```

  What is a “master node”, and what does it do? How do I configure a node so that it is the master?
- 16.6.8: [1378] With which operating systems can I use Cluster?
- 16.6.9: [1378] What are the hardware requirements for running MySQL Cluster?

Questions and Answers

16.6.1: Which versions of the MySQL software support Cluster? Do I have to compile from source?

Beginning with MySQL 4.1.3, MySQL Cluster is supported in all `MySQL-Max` server binaries in the 4.1 release series for operating systems on which MySQL Cluster is available. See Section 4.3.1, “`mysqld` — The MySQL Server”. You can determine whether your server has `NDB` support using either of the statements `SHOW VARIABLES LIKE 'have_%'` or `SHOW ENGINES`.

You can also obtain `NDB` support by compiling MySQL from source, but it is not necessary to do so simply to use MySQL Cluster. To download the latest binary, RPM, or source distribution in the MySQL 4.1 series, visit http://dev.mysql.com/downloads/mysql/4.1.html.

However, you should use MySQL NDB Cluster NDB 7.0 or 7.1 for new deployments, and if you are already using MySQL 4.1 with clustering support, to upgrade to one of these MySQL Cluster release series. For an overview of improvements made in MySQL Cluster NDB 7.0 and 7.1, see MySQL Cluster Development in MySQL Cluster NDB 7.0, and MySQL Cluster Development in MySQL Cluster NDB 7.1, respectively.

16.6.2: What does “NDB” mean?

“NDB” stands for “**N**etwork **D**ata**b**ase”. `NDB` and `NDBCLUSTER` are both names for the storage engine that enables clustering support in MySQL. Either name is equally correct; both names appear in our documentation, and either name can be used in the `ENGINE` option of a `CREATE TABLE` statement for creating a MySQL Cluster table.

16.6.3: What is the difference between using MySQL Cluster *vs* using MySQL replication?

In traditional MySQL replication, a master MySQL server updates one or more slaves. Transactions are committed sequentially, and a slow transaction can cause the slave to lag behind the master. This means that if the master fails, it is possible that the slave might not have recorded the last few transactions. If a transaction-safe engine such as `InnoDB` is being used, a transaction will either be complete on the slave or not applied at all, but replication does not guarantee that all data on the master and the slave will be consistent at all times. In MySQL Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is *asynchronous*, MySQL Cluster is *synchronous*.

We have implemented (asynchronous) replication for Cluster in MySQL 5.1 and later. *MySQL Cluster Replication* (also sometimes known as “geo-replication”) includes the capability to replicate both between two MySQL Clusters, and from a MySQL Cluster to a non-Cluster MySQL server. However, we do *not* plan to backport this functionality to MySQL 4.1. See MySQL Cluster Replication.

16.6.4: Do I need to do any special networking to run MySQL Cluster? How do computers in a cluster communicate?

MySQL Cluster is intended to be used in a high-bandwidth environment, with computers connecting using TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for MySQL Cluster include a typical 100-megabit Ethernet network or the equivalent. Use gigabit Ethernet whenever available.

The faster SCI protocol is also supported, but requires special hardware. See Section 15.3.5, “Using High-Speed Interconnects with MySQL Cluster”, for more information about SCI.

16.6.5: How many computers do I need to run a MySQL Cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum *recommended* number of computers in a MySQL Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

16.6.6: What do the different computers do in a MySQL Cluster?

A MySQL Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as *nodes*, and a computer housing a cluster node is sometimes referred to as a *cluster host*. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node.** This node provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application `ndb_mgmd`; the management client used to control MySQL Cluster is `ndb_mgm`. See Section 15.4.2, “`ndb_mgmd` — The MySQL Cluster Management Server Daemon”, and Section 15.4.3, “`ndb_mgm` — The MySQL Cluster Management Client”, for information about these programs.

- **Data node.** This type of node stores and replicates data. Data node functionality is handled by instances of the `NDB` data node process `ndbd`. For more information, see Section 15.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”.

- **SQL node.** This is simply an instance of MySQL Server (`mysqld`) that is built with support for the `NDBCLUSTER` storage engine and started with the `--ndb-cluster` option to enable the engine and the `--ndb-connectstring` option to enable it to connect to a MySQL Cluster management server. For more about these options, see Section 15.3.4.2, “`mysqld` Command Options for MySQL Cluster”.

Note

An *API node* is any application that makes direct use of Cluster data nodes for data storage and retrieval. An SQL node can thus be considered a type of API node that uses a MySQL Server to provide an SQL interface to the Cluster. You can write such applications (that do not depend on a MySQL Server) using the NDB API, which supplies a direct, object-oriented transaction and scanning interface to MySQL Cluster data; see MySQL Cluster API Overview: The NDB API, for more information.

16.6.7: When I run the `SHOW` command in the MySQL Cluster management client, I see a line of output that looks like this:

```
id=2    @10.100.10.32  (Version: 4.1.25, Nodegroup: 0, Master)
```

What is a “master node”, and what does it do? How do I configure a node so that it is the master?

The simplest answer is, “It's not something you can control, and it's nothing that you need to worry about in any case, unless you're a software engineer writing or analyzing the MySQL Cluster source code”.

If you don't find that answer satisfactory, here's a longer and more technical version:

A number of mechanisms in MySQL Cluster require distributed coordination among the data nodes. These distributed algorithms and protocols include global checkpointing, DDL (schema) changes, and node restart handling. To make this coordination simpler, the data nodes “elect” one of their number to be a “master”. There is no user-facing mechanism for influencing this selection, which is is completely automatic; the fact that it *is* automatic is a key part of MySQL Cluster's internal architecture.

When a node acts as a master for any of these mechanisms, it is usually the point of coordination for the activity, and the other nodes act as “servants”, carrying out their parts of the activity as directed by the master. If the node acting as master fails, then the remaining nodes elect a new master. Tasks in progress that were being coordinated by the old master may either fail or be continued by the new master, depending on the actual mechanism involved.

It is possible for some of these different mechanisms and protocols to have different master nodes, but in general the same master is chosen for all of them. The node indicated as the master in the output of `SHOW` in the management client is actually the `DICT` master (see The `DBDICT` Block, in the *MySQL Cluster API Developer Guide*, for more information), responsible for coordinating DDL and metadata activity.

MySQL Cluster is designed in such a way that the choice of master has no discernable effect outside the cluster itself. For example, the current master does not have significantly higher CPU or resource usage than the other data nodes, and failure of the master should not have a significantly different impact on the cluster than the failure of any other data node.

16.6.8: With which operating systems can I use Cluster?

MySQL Cluster is supported on most Unix-like operating systems, including Linux, Mac OS X, and Solaris. Beginning with MySQL Cluster NDB 7.1.3, MySQL Cluster is also supported in production on Microsoft Windows operating systems.

Important

We do not intend to provide any level of support on Windows for MySQL Cluster in MySQL 4.1; you must use MySQL Cluster NDB 7.1.3 or later to obtain GA-level support for MySQL Cluster in a Windows environment. See MySQL Cluster Development in MySQL Cluster NDB 7.1, for more information.

For more detailed information concerning the level of support which is offered for MySQL Cluster on various operating system versions, operating system distributions, and hardware platforms, please refer to `http://www.mysql.com/support/supportedplatforms/cluster.html`.

16.6.9: What are the hardware requirements for running MySQL Cluster?

MySQL Cluster should run on any platform for which `NDB`-enabled binaries are available. For data nodes and API nodes, faster CPUs and more memory are likely to improve performance, and 64-bit CPUs are likely to be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see *How much RAM do I Need?* for more information). For a computer which is used only for running the MySQL Cluster management server, the requirements are minimal; a common desktop PC (or the equivalent) is generally sufficient for this task. Nodes can communicate through the standard TCP/IP network and hardware. They can also use the high-speed SCI protocol; however, special networking hardware and software are required to use SCI (see Section 15.3.5, “Using High-Speed Interconnects with MySQL Cluster”).

16.6.10: How much RAM do I need to use MySQL Cluster? Is it possible to use disk memory at all?

In MySQL 4.1, Cluster is in-memory only. This means that all table data (including indexes) is stored in RAM. Therefore, if your data takes up 1 GB of space and you want to replicate it once in the cluster, you need 2 GB of memory to do so (1 GB per replica). This is in addition to the memory required by the operating system and any applications running on the cluster computers.

If a data node's memory usage exceeds what is available in RAM, then the system will attempt to use swap space up to the limit set for `DataMemory`. However, this will at best result in severely degraded performance, and may cause the node to be dropped due to slow response time (missed heartbeats). We do not recommend on relying on disk swapping in a production environment for this reason. In any case, once the `DataMemory` limit is reached, any operations requiring additional memory (such as inserts) will fail.

We have implemented disk data storage for MySQL Cluster in MySQL 5.1 and later but we have no plans to add this capability in MySQL 4.1. See MySQL Cluster Disk Data Tables, for more information.

You can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

```
(SizeofDatabase × NumberOfReplicas × 1.1 ) / NumberOfDataNodes
```

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see Section 10.5, “Data Type Storage Requirements”, for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an `NDBCLUSTER` table requires 21–25 bytes per record. These indexes use `IndexMemory`.
- Each ordered index requires 10 bytes storage per record, using `DataMemory`.
- Creating a primary key or unique index also creates an ordered index, unless this index is created with `USING HASH`. In other words:
  - A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.
  - However, if the primary key or unique index is created with `USING HASH`, then it requires only 21 to 25 bytes per record.

Note that creating MySQL Cluster tables with `USING HASH` for all primary keys and unique indexes will generally cause table updates to run more quickly—in some cases by a much as 20 to 30 percent faster than updates on tables where `USING HASH` was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the `ndb_size.pl` utility which is available in recent MySQL 4.1 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on how much space that database would require if it used the `NDBCLUSTER` storage engine. For more information, see Section 15.4.18, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”.

It is especially important to keep in mind that *every MySQL Cluster table must have a primary key*. The `NDB` storage engine creates a primary key automatically if none is defined; this primary key is created without `USING HASH`.

There is no easy way to determine exactly how much memory is being used for storage of Cluster indexes at any given time; however, warnings are written to the Cluster log when 80% of available `DataMemory` or `IndexMemory` is in use, and again when use reaches 85%, 90%, and so on.

16.6.11: What file systems can I use with MySQL Cluster? What about network file systems or network shares?

Generally, any file system that is native to the host operating system should work well with MySQL Cluster. If you find that a given file system works particularly well (or not so especially well) with MySQL Cluster, we invite you to discuss your findings in the MySQL Cluster Forums.

We do not test MySQL Cluster with FAT or VFAT file systems on Linux. Because of this, and due to the fact that these are not very useful for any purpose other than sharing disk partitions between Linux and Windows operating systems on multi-boot computers, we do not recommend their use with MySQL Cluster.

MySQL Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network file systems is not supported for MySQL Cluster. This also applies to shared storage devices such as SANs.

16.6.12: Can I run MySQL Cluster nodes inside virtual machines (such as those created by VMWare, Parallels, or Xen)?

This is possible but not recommended for a production environment.

We have found that running MySQL Cluster processes inside a virtual machine can give rise to issues with timing and disk subsystems that have a strong negative impact on the operation of the cluster. The behavior of the cluster is often unpredictable in these cases.

If an issue can be reproduced outside the virtual environment, then we may be able to provide assistance. Otherwise, we cannot support it at this time.

16.6.13: I am trying to populate a MySQL Cluster database. The loading process terminates prematurely and I get an error message like this one: `ERROR 1114: The table 'my_cluster_table' is full` Why is this happening?

The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by the NDB storage engine and automatically created in the event that the table definition does not include the definition of a primary key*.

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. For example, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote at most 1GB to MySQL Cluster data and indexes.

16.6.14: MySQL Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is *very* unlikely that a cluster would perform reliably under such conditions, as MySQL Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet —preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in a MySQL Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you would with any other instance of the MySQL server.) For more information, see Section 15.5.9, “MySQL Cluster Security Issues”.

16.6.15: Do I have to learn a new programming or query language to use MySQL Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL statements are required for the following operations:

- Creating, altering, and dropping tables
- Inserting, updating, and deleting table data
- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up a MySQL Cluster—see Section 15.3.2, “MySQL Cluster Configuration Files”, for information about these.

A few simple commands are used in the MySQL Cluster management client (`ndb_mgm`) for tasks such as starting and stopping cluster nodes. See Section 15.5.2, “Commands in the MySQL Cluster Management Client”.

16.6.16: How do I find out what an error or warning message means when using MySQL Cluster?

There are two ways in which this can be done:

-
- From a system shell prompt, use `perror --ndb error_code`.

16.6.17: Is MySQL Cluster transaction-safe? What isolation levels are supported?

Yes. For tables created with the `NDB` storage engine, transactions are supported. Currently, MySQL Cluster supports only the `READ COMMITTED` [975] transaction isolation level.

16.6.18: What storage engines are supported by MySQL Cluster?

Clustering with MySQL is supported only by the `NDB` storage engine. That is, in order for a table to be shared between nodes in a MySQL Cluster, the table must be created using `ENGINE=NDB` (or the equivalent option `ENGINE=NDBCLUSTER`).

It is possible to create tables using other storage engines (such as `MyISAM` or `InnoDB`) on a MySQL server being used with a MySQL Cluster, but these non-`NDB` tables do *not* participate in clustering; each such table is strictly local to the individual MySQL server instance on which it is created.

16.6.19: In the event of a catastrophic failure—say, for instance, the whole city loses power and my UPS fails—would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

16.6.20: Is it possible to use `FULLTEXT` indexes with MySQL Cluster?

`FULLTEXT` indexing is not supported by any storage engine other than `MyISAM`. We are working to add this capability to MySQL Cluster tables in a future release.

16.6.21: Can I run multiple nodes on a single computer?

It is possible but not advisable. One of the chief reasons to run a cluster is to provide redundancy. To obtain the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. Given that MySQL

Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 300 MHz Pentium or equivalent CPU and sufficient RAM for the operating system, plus a small amount of overhead for the `ndb_mgmd` and `ndb_mgm` processes.

It is acceptable to run multiple cluster data nodes on a single host for learning about MySQL Cluster, or for testing purposes; however, this is not generally supported for production use.

16.6.22: Can I add data nodes to a MySQL Cluster without restarting it?

Not in MySQL 4.1. While a rolling restart is all that is required for adding new management or API nodes to a MySQL Cluster (see Section 15.2.6.1, “Performing a Rolling Restart of a MySQL Cluster”), adding data nodes is more complex, and requires the following steps:

1. Make a complete backup of all Cluster data.
2. Completely shut down the cluster and all cluster node processes.
3. Restart the cluster, using the `--initial` startup option for all instances of `ndbd`.

Warning

Never use the `--initial` when starting `ndbd` except when necessary to clear the data node file system. See Section 15.4.1, “`ndbd` — The MySQL Cluster Data Node Daemon”, for information about when this is required.

4. Restore all cluster data from the backup.

Note

Beginning with MySQL Cluster NDB 6.4, it is possible to add new data nodes to a running MySQL Cluster without taking it offline. For more information, see Adding MySQL Cluster Data Nodes Online. However, we do not plan to add this capability in MySQL 4.1.

16.6.23: Are there any limitations that I should be aware of when using MySQL Cluster?

Limitations on `NDB` tables in MySQL 4.1 include the following:

- Temporary tables are not supported; a `CREATE TEMPORARY TABLE` statement using `ENGINE=NDB` or `ENGINE=NDBCLUSTER` fails with an error.
- `FULLTEXT` indexes are not supported.
- Index prefixes are not supported. Only complete columns may be indexed.
- In MySQL 4.1, MySQL Cluster does not support spatial data types or spatial indexes. See Chapter 16, *Spatial Extensions*.
- Only complete rollbacks for transactions are supported. Partial rollbacks and rollbacks to savepoints are not supported. A failed insert due to a duplicate key or similar error causes a transaction to abort; when this occurs, you must issue an explicit `ROLLBACK` and retry the transaction.
- The maximum number of attributes permitted per table is 128, and attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.

- The maximum size for a table row is 8 kilobytes, not counting `BLOB` values. There is no set limit for the number of rows per table. Table size limits depend on a number of factors, in particular on the amount of RAM available to each data node.
- The `NDB` engine does not support foreign key constraints. As with `MyISAM` tables, if these are specified in a `CREATE TABLE` or `ALTER TABLE` statement, they are ignored.

For a complete listing of limitations in MySQL Cluster, see Section 15.1.4, “Known Limitations of MySQL Cluster”.

16.6.24: How do I import an existing MySQL database into a MySQL Cluster?

You can import databases into MySQL Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ, the only other special requirement is that any tables to be included in the cluster must use the `NDB` storage engine. This means that the tables must be created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`.

It is also possible to convert existing tables that use other storage engines to `NDBCLUSTER` using one or more `ALTER TABLE` statement. However, the definition of the table must be compatible with the `NDBCLUSTER` storage engine prior to making the conversion. In MySQL 4.1, an additional workaround is also required; see Section 15.1.4, “Known Limitations of MySQL Cluster”, for details.

16.6.25: How do cluster nodes communicate with one another?

Cluster nodes can communicate through any of three different transport mechanisms: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See Section 15.3.5, “Using High-Speed Interconnects with MySQL Cluster”, for more about using SCI as a transport mechanism for MySQL Cluster.

16.6.26: What is an *arbitrator*?

If one or more data nodes in a cluster fail, it is possible that not all cluster data nodes will be able to “see” one another. In fact, it is possible that two sets of data nodes might become isolated from one another in a network partitioning, also known as a “split-brain” scenario. This type of situation is undesirable because each set of data nodes tries to behave as though it is the entire cluster. An arbitrator is required to decide between the competing sets of data nodes.

When all data nodes in at least one node group are alive, network partitioning is not an issue, because no single subset of the cluster can form a functional cluster on its own. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain” scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the `ArbitrationRank` configuration parameter for MySQL Server and management server nodes. (See Section 15.3.2.4, “Defining a MySQL Cluster Management Server”, for details.)

The role of arbitrator does not in and of itself impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

16.6.27: What data types are supported by MySQL Cluster?

In MySQL 4.1, MySQL Cluster supports all of the usual MySQL data types, except for those associated with MySQL's spatial extensions. (Spatial data types and spatial indexes are supported only by `MyISAM`;

see Chapter 16, *Spatial Extensions*, for more information.) In addition, there are some differences with regard to indexes when used with NDB tables.

Note

In MySQL 4.1, MySQL Cluster tables (that is, tables created with ENGINE=NDB or ENGINE=NDBCLUSTER) have only fixed-width rows. This means that (for example) each record containing a VARCHAR(255) column will require space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein. This issue is fixed in MySQL 5.1 and later; however, we do not plan to backport this functionality to MySQL 4.1.

See Section 15.1.4, "Known Limitations of MySQL Cluster", for more information about these issues.

16.6.28: How do I start and stop MySQL Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node, using the ndb_mgmd command.

   You must include the -f or --config-file [ ] option to tell the management node where its configuration file can be found.

2. Start each data node with the ndbd command.

   Each data node must be started with the -c [1332] or --connect-string [1332] option so that the data node knows how to connect to the management server.

3. Start each MySQL Server (SQL node) using your preferred startup script, such as mysqld_safe.

   Each MySQL Server must be started with the --ndbcluster [1301] and --ndb-connectstring [1301] options. These options cause mysqld to enable NDBCLUSTER storage engine support and how to connect to the management server.

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine—a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the NDB management client ndb_mgm on the machine housing the management node and issuing the SHOW or ALL STATUS command.

To shut down a running cluster, issue the command SHUTDOWN in the management client. Alternatively, you may enter the following command in a system shell:

```
shell> ndb_mgm -e "SHUTDOWN"
```

(The quotation marks in this example are optional, since there are no spaces in the command string following the -e option; in addition, the SHUTDOWN command, like other management client commands, is not case-sensitive.)

Either of these commands causes the ndb_mgm, ndb_mgm, and any ndbd processes to terminate gracefully. MySQL servers running as SQL nodes can be stopped using mysqladmin shutdown.

For more information, see Section 15.5.2, "Commands in the MySQL Cluster Management Client", and Section 15.2.5, "Safe Shutdown and Restart of MySQL Cluster".

16.6.29: What happens to MySQL Cluster data when the cluster is shut down?

The data that was held in memory by the cluster's data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

16.6.30: Is it a good idea to have more than one management node for a MySQL Cluster?

It can be helpful as a fail-safe. Only one management node controls the cluster at any given time, but it is possible to configure one management node as primary, and one or more additional management nodes to take over in the event that the primary management node fails.

See Section 15.3.2, “MySQL Cluster Configuration Files”, for information on how to configure MySQL Cluster management nodes.

16.6.31: Can I mix different kinds of hardware and operating systems in one MySQL Cluster?

Yes, as long as all machines and operating systems have the same “endianness” (all big-endian or all little-endian). We are working to overcome this limitation in a future MySQL Cluster release.

It is also possible to use software from different MySQL Cluster releases on different nodes. However, we support this only as part of a rolling upgrade procedure (see Section 15.2.6.1, “Performing a Rolling Restart of a MySQL Cluster”).

16.6.32: Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of `mysqld` must use a different TCP/IP port. However, in MySQL 4.1, running more than one cluster node of a given type per machine is generally not encouraged or supported for production use.

We also advise against running data nodes and SQL nodes together on the same host, since the `ndbd` and `mysqld` processes may compete for memory.

16.6.33: Can I use host names with MySQL Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires “five nines” availability, you should use fixed (numeric) IP addresses, since making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional potential points of failure.

16.6.34: How do I handle MySQL users in a MySQL Cluster having multiple MySQL servers?

MySQL user accounts and privileges are *not* automatically propagated between different MySQL servers accessing the same MySQL Cluster. Therefore, you must make sure that these are copied between the SQL nodes yourself. You can do this manually, or automate the task with scripts.

Warning

Do not attempt to work around this issue by converting the MySQL system tables to use the `NDBCLUSTER` storage engine. Only the `MyISAM` storage engine is supported for these tables.

16.6.35: How do I continue to send queries in the event that one of the SQL nodes fails?

MySQL Cluster does not provide any sort of automatic failover between SQL nodes. Your application must be prepared to handlethe loss of SQL nodes and to fail over between them.

16.6.36: How do I back up and restore a MySQL Cluster?

You can use the NDB native backup and restore functionality in the MySQL Cluster management client and the `ndb_restore` program. See Section 15.5.3, “Online Backup of MySQL Cluster”, and Section 15.4.14, “`ndb_restore` — Restore a MySQL Cluster Backup”.

You can also use the traditional functionality provided for this purpose in `mysqldump` and the MySQL server. See Section 4.5.4, “`mysqldump` — A Database Backup Program”, for more information.

16.6.37: What is an “angel process”?

This process monitors and, if necessary, attempts to restart the data node process. If you check the list of active processes on your system after starting `ndbd`, you can see that there are actually 2 processes running by that name, as shown here (we omit the output from `ndb_mgmd` and `ndbd` for brevity):

```
shell> ./ndb_mgmd

shell> ps aux | grep ndb
me      23002  0.0  0.0 122948  3104 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23025  0.0  0.0   5284   820 pts/2    S+   14:14   0:00 grep ndb

shell> ./ndbd -c 127.0.0.1 --initial

shell> ps aux | grep ndb
me      23002  0.0  0.0 123080  3356 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23096  0.0  0.0  35876  2036 ?        Ss   14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23097  1.0  2.4 524116 91096 ?        Sl   14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23168  0.0  0.0   5284   812 pts/2    R+   14:15   0:00 grep ndb
```

The `ndbd` process showing 0 memory and CPU usage is the angel process. It actually does use a very small amount of each, of course. It simply checks to see if the main `ndbd` process (the primary data node process that actually handles the data) is running. If permitted to do so (for example, if the `StopOnError` configuration parameter is set to false—see Section 15.3.3.1, “MySQL Cluster Data Node Configuration Parameters”), the angel process tries to restart the primary data node process.

Chapter 16 Spatial Extensions

Table of Contents

MySQL 4.1 introduces spatial extensions to enable the generation, storage, and analysis of geographic features. Currently, these features are available for `MyISAM` tables only.

This chapter covers the following topics:

- The basis of these spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data

- MySQL differences from the OpenGIS specification

Additional Resources

- The Open Geospatial Consortium publishes the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at http://www.opengis.org/docs/99-049.pdf.
- If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: http://forums.mysql.com/list.php?23.

16.1 Introduction to MySQL Spatial Support

MySQL implements spatial extensions following the specification of the Open Geospatial Consortium (OGC). This is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The OGC maintains a Web site at http://www.opengis.org/.

In 1997, the Open Geospatial Consortium published the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at http://www.opengis.org/docs/99-049.pdf. It contains additional information relevant to this chapter.

MySQL implements a subset of the **SQL with Geometry Types** environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describe a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, town district, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

This chapter uses all of these terms synonymously: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. Here, the term most commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

16.2 The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.

- It belongs to some geometry class.

16.2.1 The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- `Geometry` (noninstantiable)
  - `Point` (instantiable)
  - `Curve` (noninstantiable)
    - `LineString` (instantiable)
      - `Line`
      - `LinearRing`
  - `Surface` (noninstantiable)
    - `Polygon` (instantiable)
  - `GeometryCollection` (instantiable)
    - `MultiPoint` (instantiable)
    - `MultiCurve` (noninstantiable)
      - `MultiLineString` (instantiable)
    - `MultiSurface` (noninstantiable)
      - `MultiPolygon` (instantiable)

It is not possible to create objects in noninstantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It is an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.
- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.
- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.
- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

Geometry, Curve, Surface, MultiCurve, and MultiSurface are defined as noninstantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString, and MultiPolygon are instantiable classes.

16.2.2 Class Geometry

Geometry is the root class of the hierarchy. It is a noninstantiable class but has a number of properties that are common to all geometry values created from any of the Geometry subclasses. These properties are described in the following list. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

  In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.
- Its **coordinates** in its Spatial Reference System, represented as double-precision (eight-byte) numbers. All nonempty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

  Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geocentric** system (coordinates on the Earth's surface) are different things.
- Its **interior**, **boundary**, and **exterior**.

  Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.
- Its **MBR** (Minimum Bounding Rectangle), or Envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

  ```
  ((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```
- Whether the value is **simple** or **nonsimple**. Geometry values of types (LineString, MultiPoint, MultiLineString) are either simple or nonsimple. Each type determines its own assertions for being simple or nonsimple.
- Whether the value is **closed** or **not closed**. Geometry values of types (LineString, MultiString) are either closed or not closed. Each type determines its own assertions for being closed or not closed.
- Whether the value is **empty** or **nonempty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a NULL value). An empty geometry is defined to be always simple and has an area of 0.
- Its **dimension**. A geometry can have a dimension of –1, 0, 1, or 2:

- –1 for an empty geometry.
- 0 for a geometry with no length and no area.
- 1 for a geometry with nonzero length and zero area.
- 2 for a geometry with nonzero area.

`Point` objects have a dimension of zero. `LineString` objects have a dimension of 1. `Polygon` objects have a dimension of 2. The dimensions of `MultiPoint`, `MultiLineString`, and `MultiPolygon` objects are the same as the dimensions of the elements they consist of.

16.2.3 Class `Point`

A `Point` is a geometry that represents a single location in coordinate space.

`Point` Examples

- Imagine a large-scale map of the world with many cities. A `Point` object could represent each city.
- On a city map, a `Point` object could represent a bus stop.

`Point` Properties

- X-coordinate value.
- Y-coordinate value.
- `Point` is defined as a zero-dimensional geometry.
- The boundary of a `Point` is the empty set.

16.2.4 Class `Curve`

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` define the type of interpolation between points. `Curve` is a noninstantiable class.

`Curve` Properties

- A `Curve` has the coordinates of its points.
- A `Curve` is defined as a one-dimensional geometry.
- A `Curve` is simple if it does not pass through the same point twice.
- A `Curve` is closed if its start point is equal to its endpoint.
- The boundary of a closed `Curve` is empty.
- The boundary of a nonclosed `Curve` consists of its two endpoints.
- A `Curve` that is simple and closed is a `LinearRing`.

16.2.5 Class `LineString`

A `LineString` is a `Curve` with linear interpolation between points.

`LineString` Examples

- On a world map, LineString objects could represent rivers.
- In a city map, LineString objects could represent streets.

LineString Properties

- A LineString has coordinates of segments, defined by each consecutive pair of points.
- A LineString is a Line if it consists of exactly two points.
- A LineString is a LinearRing if it is both closed and simple.

16.2.6 Class Surface

A Surface is a two-dimensional geometry. It is a noninstantiable class. Its only instantiable subclass is Polygon.

Surface Properties

- A Surface is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple Surface as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple Surface is the set of closed curves corresponding to its exterior and interior boundaries.

16.2.7 Class Polygon

A Polygon is a planar Surface representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the Polygon.

Polygon Examples

- On a region map, Polygon objects could represent forests, districts, and so on.

Polygon Assertions

- The boundary of a Polygon consists of a set of LinearRing objects (that is, LineString objects that are both simple and closed) that make up its exterior and interior boundaries.
- A Polygon has no rings that cross. The rings in the boundary of a Polygon may intersect at a Point, but only as a tangent.
- A Polygon has no lines, spikes, or punctures.
- A Polygon has an interior that is a connected point set.
- A Polygon may have holes. The exterior of a Polygon with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a Polygon a simple geometry.

16.2.8 Class GeometryCollection

A GeometryCollection is a geometry that is a collection of one or more geometries of any class.

All the elements in a GeometryCollection must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a GeometryCollection, although the subclasses of GeometryCollection described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a MultiPoint may contain only Point elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

16.2.9 Class MultiPoint

A MultiPoint is a geometry collection composed of Point elements. The points are not connected or ordered in any way.

MultiPoint Examples

- On a world map, a MultiPoint could represent a chain of small islands.
- On a city map, a MultiPoint could represent the outlets for a ticket office.

MultiPoint Properties

- A MultiPoint is a zero-dimensional geometry.
- A MultiPoint is simple if no two of its Point values are equal (have identical coordinate values).
- The boundary of a MultiPoint is the empty set.

16.2.10 Class MultiCurve

A MultiCurve is a geometry collection composed of Curve elements. MultiCurve is a noninstantiable class.

MultiCurve Properties

- A MultiCurve is a one-dimensional geometry.
- A MultiCurve is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A MultiCurve boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a MultiCurve if it is in the boundaries of an odd number of MultiCurve elements.
- A MultiCurve is closed if all of its elements are closed.
- The boundary of a closed MultiCurve is always empty.

16.2.11 Class MultiLineString

A MultiLineString is a MultiCurve geometry collection composed of LineString elements.

MultiLineString Examples

- On a region map, a MultiLineString could represent a river system or a highway system.

16.2.12 Class MultiSurface

A MultiSurface is a geometry collection composed of surface elements. MultiSurface is a noninstantiable class. Its only instantiable subclass is MultiPolygon.

MultiSurface Assertions

- Two MultiSurface surfaces have no interiors that intersect.
- Two MultiSurface elements have boundaries that intersect at most at a finite number of points.

16.2.13 Class MultiPolygon

A MultiPolygon is a MultiSurface object composed of Polygon elements.

MultiPolygon Examples

- On a region map, a MultiPolygon could represent a system of lakes.

MultiPolygon Assertions

- A MultiPolygon has no two Polygon elements with interiors that intersect.
- A MultiPolygon has no two Polygon elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A MultiPolygon may not have cut lines, spikes, or punctures. A MultiPolygon is a regular, closed point set.
- A MultiPolygon that has more than one Polygon has an interior that is not connected. The number of connected components of the interior of a MultiPolygon is equal to the number of Polygon values in the MultiPolygon.

MultiPolygon Properties

- A MultiPolygon is a two-dimensional geometry.
- A MultiPolygon boundary is a set of closed curves (LineString values) corresponding to the boundaries of its Polygon elements.
- Each Curve in the boundary of the MultiPolygon is in the boundary of exactly one Polygon element.
- Every Curve in the boundary of an Polygon element is in the boundary of the MultiPolygon.

16.3 Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to represent geometry objects in queries. They are:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

16.3.1 Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form. For a Backus-Naur grammar that specifies the formal production rules for writing WKT values, see the OpenGIS specification document referenced in Chapter 16, *Spatial Extensions*.

Examples of WKT representations of geometry objects:

- A `Point`:

```
POINT(15 20)
```

  Note that point coordinates are specified with no separating comma. This differs from the syntax for the SQL `POINT()` [1399] function, which requires a comma between the coordinates. Take care to use the syntax appropriate to the context of a given spatial operation. For example, the following statements both extract the X-coordinate from a `Point` object. The first produces the object directly using the `POINT()` [1399] function. The second uses a WKT representation converted to a `Point` with `GeomFromText()`.

```
mysql> SELECT X(POINT(15, 20));
+-----------------+
| X(POINT(15, 20)) |
+-----------------+
|              15 |
+-----------------+

mysql> SELECT X(GeomFromText('POINT(15 20)'));
+---------------------------------+
| X(GeomFromText('POINT(15 20)')) |
+---------------------------------+
|                              15 |
+---------------------------------+
```

- A `LineString` with four points:

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

  Note that point coordinate pairs are separated by commas.

- A `Polygon` with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A `MultiPoint` with three `Point` values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A `MultiLineString` with two `LineString` values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A `MultiPolygon` with two `Polygon` values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A `GeometryCollection` consisting of two `Point` values and one `LineString`:

```
```

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

16.3.2 Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation for geometric values is defined by the OpenGIS specification. It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB is used to exchange geometry data as binary streams represented by `BLOB` values containing geometric WKB information.

WKB uses one-byte unsigned integers, four-byte unsigned integers, and eight-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to `POINT(1 1)` consists of this sequence of 21 bytes (each represented here by two hex digits):

```
0101000000000000000000F03F000000000000F03F
```

The sequence may be broken down into these components:

```
Byte order : 01
WKB type   : 01000000
X          : 000000000000F03F
Y          : 000000000000F03F
```

Component representation is as follows:

- The byte order may be either 1 or 0 to indicate little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.
- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`.
- A `Point` value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values are represented by more complex data structures, as detailed in the OpenGIS specification.

16.4 Creating a Spatially Enabled MySQL Database

This section describes the data types you can use for representing spatial data in MySQL, and the functions available for creating and retrieving spatial values.

16.4.1 MySQL Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINESTRING`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types (`POINT`, `LINESTRING`, and `POLYGON`) restrict their values to a particular geometry type.

The other data types hold collections of values:

- `MULTIPOINT`
- `MULTILINESTRING`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types (`MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, and `GEOMETRYCOLLECTION`) restrict collection members to those having a particular geometry type.

16.4.2 Creating Spatial Values

This section describes how to create spatial values using Well-Known Text and Well-Known Binary functions that are defined in the OpenGIS standard, and using MySQL-specific functions.

16.4.2.1 Creating Geometry Values Using WKT Functions

MySQL provides a number of functions that take as arguments a Well-Known Text representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` [1397] accepts a WKT of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromText(`*`wkt`*`[,`*`srid`*`])` [1397], `GeometryCollectionFromText(`*`wkt`*`[,`*`srid`*`])` [1397]

  Constructs a `GEOMETRYCOLLECTION` value using its WKT representation and SRID.

- `GeomFromText(`*`wkt`*`[,`*`srid`*`])` [1397], `GeometryFromText(`*`wkt`*`[,`*`srid`*`])` [1397]

  Constructs a geometry value of any type using its WKT representation and SRID.

- `LineFromText(`*`wkt`*`[,`*`srid`*`])` [1397], `LineStringFromText(`*`wkt`*`[,`*`srid`*`])` [1397]

  Constructs a `LINESTRING` value using its WKT representation and SRID.

- `MLineFromText(`*`wkt`*`[,`*`srid`*`])` [1397], `MultiLineStringFromText(`*`wkt`*`[,`*`srid`*`])` [1397]

  Constructs a `MULTILINESTRING` value using its WKT representation and SRID.

- `MPointFromText(`*`wkt`*`[,`*`srid`*`])` [1397], `MultiPointFromText(`*`wkt`*`[,`*`srid`*`])` [1397]

  Constructs a `MULTIPOINT` value using its WKT representation and SRID.

- `MPolyFromText(`*`wkt`*`[,`*`srid`*`])` [1397], `MultiPolygonFromText(`*`wkt`*`[,`*`srid`*`])` [1397]

  Constructs a `MULTIPOLYGON` value using its WKT representation and SRID.

- `PointFromText(`*`wkt`*`[,`*`srid`*`])` [1397]

  Constructs a `POINT` value using its WKT representation and SRID.

- `PolyFromText(wkt[,srid])` [1398], `PolygonFromText(wkt[,srid])` [1398]

  Constructs a `POLYGON` value using its WKT representation and SRID.

The OpenGIS specification also defines the following optional functions, which MySQL does not implement. These functions construct `Polygon` or `MultiPolygon` values based on the WKT representation of a collection of rings or closed `LineString` values. These values may intersect.

- `BdMPolyFromText(wkt,srid)` [1398]

  Constructs a `MultiPolygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.

- `BdPolyFromText(wkt,srid)` [1398]

  Constructs a `Polygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.

16.4.2.2 Creating Geometry Values Using WKB Functions

MySQL provides a number of functions that take as arguments a `BLOB` containing a Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromWKB()` [1398] accepts a WKB of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromWKB(wkb[,srid])` [1398], `GeometryCollectionFromWKB(wkb[,srid])` [1398]

  Constructs a `GEOMETRYCOLLECTION` value using its WKB representation and SRID.

- `GeomFromWKB(wkb[,srid])` [1398], `GeometryFromWKB(wkb[,srid])` [1398]

  Constructs a geometry value of any type using its WKB representation and SRID.

- `LineFromWKB(wkb[,srid])` [1398], `LineStringFromWKB(wkb[,srid])` [1398]

  Constructs a `LINESTRING` value using its WKB representation and SRID.

- `MLineFromWKB(wkb[,srid])` [1398], `MultiLineStringFromWKB(wkb[,srid])` [1398]

  Constructs a `MULTILINESTRING` value using its WKB representation and SRID.

- `MPointFromWKB(wkb[,srid])` [1398], `MultiPointFromWKB(wkb[,srid])` [1398]

  Constructs a `MULTIPOINT` value using its WKB representation and SRID.

- `MPolyFromWKB(wkb[,srid])` [1398], `MultiPolygonFromWKB(wkb[,srid])` [1398]

  Constructs a `MULTIPOLYGON` value using its WKB representation and SRID.

- `PointFromWKB(wkb[,srid])` [1398]

  Constructs a `POINT` value using its WKB representation and SRID.

- `PolyFromWKB(wkb[,srid])` [1398], `PolygonFromWKB(wkb[,srid])` [1398]

  Constructs a `POLYGON` value using its WKB representation and SRID.

The OpenGIS specification also describes optional functions for constructing `Polygon` or `MultiPolygon` values based on the WKB representation of a collection of rings or closed `LineString` values. These values may intersect. MySQL does not implement these functions:

- `BdMPolyFromWKB(wkb,srid)` [1399]

  Constructs a `MultiPolygon` value from a `MultiLineString` value in WKB format containing an arbitrary collection of closed `LineString` values.

- `BdPolyFromWKB(wkb,srid)` [1399]

  Constructs a `Polygon` value from a `MultiLineString` value in WKB format containing an arbitrary collection of closed `LineString` values.

16.4.2.3 Creating Geometry Values Using MySQL-Specific Functions

MySQL provides a set of useful nonstandard functions for creating geometry WKB representations. The functions described in this section are MySQL extensions to the OpenGIS specification. The results of these functions are `BLOB` values containing WKB representations of geometry values with no SRID. The results of these functions can be substituted as the first argument for any function in the `GeomFromWKB()` [1398] function family.

- `GeometryCollection(g1,g2,...)` [1399]

  Constructs a WKB `GeometryCollection`. If any argument is not a well-formed WKB representation of a geometry, the return value is `NULL`.

- `LineString(pt1,pt2,...)` [1399]

  Constructs a WKB `LineString` value from a number of WKB `Point` arguments. If any argument is not a WKB `Point`, the return value is `NULL`. If the number of `Point` arguments is less than two, the return value is `NULL`.

- `MultiLineString(ls1,ls2,...)` [1399]

  Constructs a WKB `MultiLineString` value using WKB `LineString` arguments. If any argument is not a WKB `LineString`, the return value is `NULL`.

- `MultiPoint(pt1,pt2,...)` [1399]

  Constructs a WKB `MultiPoint` value using WKB `Point` arguments. If any argument is not a WKB `Point`, the return value is `NULL`.

- `MultiPolygon(poly1,poly2,...)` [1399]

  Constructs a WKB `MultiPolygon` value from a set of WKB `Polygon` arguments. If any argument is not a WKB `Polygon`, the return value is `NULL`.

- `Point(x,y)` [1399]

  Constructs a WKB `Point` using its coordinates.

- `Polygon(ls1,ls2,...)` [1399]

  Constructs a WKB `Polygon` value from a number of WKB `LineString` arguments. If any argument does not represent the WKB of a `LinearRing` (that is, not a closed and simple `LineString`) the return value is `NULL`.

16.4.3 Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Currently, spatial columns are supported only for `MyISAM` tables.

- Use the `CREATE TABLE` statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;
```

16.4.4 Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values into internal geometry format:

- Perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples all use `GeomFromText()` [1397] to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));
```

```
SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Note that if a client application program wants to use WKB representations of geometry values, it is responsible for sending correctly formed WKB in queries to the server. However, there are several ways of satisfying this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

```
mysql> INSERT INTO geom VALUES
    -> (GeomFromWKB(0x0101000000000000000000F03F000000000000F03F));
```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

  Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See Section 17.6.6.51, “`mysql_real_escape_string()`”.

16.4.5 Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them into WKT or WKB format.

- Fetching spatial data in internal format:

  Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

- Fetching spatial data in WKT format:

  The `AsText()` [1402] function converts a geometry from internal format into a WKT string.

```
SELECT AsText(g) FROM geom;
```

- Fetching spatial data in WKB format:

  The `AsBinary()` [1402] function converts a geometry from internal format into a `BLOB` containing the WKB value.

```
SELECT AsBinary(g) FROM geom;
```

16.5 Analyzing Spatial Information

After populating spatial columns with values, you are ready to query and analyze them. MySQL provides a set of functions to perform various operations on spatial data. These functions can be grouped into four major categories according to the type of operation they perform:

- Functions that convert geometries between various formats

- Functions that provide access to qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

Spatial analysis functions can be used in many contexts, such as:

- Any interactive SQL program, such as `mysql`.
- Application programs written in any language that supports a MySQL client API

16.5.1 Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between internal format and either WKT or WKB format:

- `AsBinary(g)` [1402], `AsWKB(g)` [1402]

  Converts a value in internal geometry format to its WKB representation and returns the binary result.

```
SELECT AsBinary(g) FROM geom;
```

- `AsText(g)` [1402], `AsWKT(g)` [1402]

  Converts a value in internal geometry format to its WKT representation and returns the string result.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+--------------------------+
| AsText(GeomFromText(@g)) |
+--------------------------+
| LINESTRING(1 1,2 2,3 3)  |
+--------------------------+
```

- `GeomFromText(wkt[,srid])` [1397]

  Converts a string value from its WKT representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as `PointFromText()` [1397] and `LineFromText()` [1397]. See Section 16.4.2.1, “Creating Geometry Values Using WKT Functions”.

- `GeomFromWKB(wkb[,srid])` [1398]

  Converts a binary value from its WKB representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as `PointFromWKB()` [1398] and `LineFromWKB()` [1398]. See Section 16.4.2.2, “Creating Geometry Values Using WKB Functions”.

16.5.2 `Geometry` Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return `NULL` if the argument is of an incorrect geometry type. For example, `Area()` [1406] returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

16.5.2.1 General Geometry Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- `Dimension(g)` [1402]

Returns the inherent dimension of the geometry value *g*. The result can be –1, 0, 1, or 2. The meaning of these values is given in Section 16.2.2, “Class Geometry”.

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+---------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+
```

- `Envelope(g)` [1403]

  Returns the Minimum Bounding Rectangle (MBR) for the geometry value *g*. The result is returned as a `Polygon` value.

  The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-----------------------------------------------------+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)'))) |
+-----------------------------------------------------+
| POLYGON((1 1,2 1,2 2,1 2,1 1))                      |
+-----------------------------------------------------+
```

- `GeometryType(g)` [1403]

  Returns as a string the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable `Geometry` subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+---------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+---------------------------------------+
| POINT                                 |
+---------------------------------------+
```

- `SRID(g)` [1403]

  Returns an integer indicating the Spatial Reference System ID for the geometry value *g*.

  In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+---------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+---------------------------------------------+
|                                         101 |
+---------------------------------------------+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- `Boundary(g)` [1403]

  Returns a geometry that is the closure of the combinatorial boundary of the geometry value *g*.

- IsEmpty(*g*) [1404]

  Returns 1 if the geometry value *g* is the empty geometry, 0 if it is not empty, and –1 if the argument is NULL. If the geometry is empty, it represents the empty point set.

- IsSimple(*g*) [1404]

  Currently, this function is a placeholder and should not be used. If implemented, its behavior will be as described in the next paragraph.

  Returns 1 if the geometry value *g* has no anomalous geometric points, such as self-intersection or self-tangency. IsSimple() [1404] returns 0 if the argument is not simple, and –1 if it is NULL.

  The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple. (See Section 16.2.1, "The Geometry Class Hierarchy".)

16.5.2.2 Point Functions

A Point consists of X and Y coordinates, which may be obtained using the following functions:

- X(*p*) [1404]

  Returns the X-coordinate value for the Point object *p* as a double-precision number.

```
mysql> SELECT X(POINT(56.7, 53.34));
+----------------------+
| X(POINT(56.7, 53.34)) |
+----------------------+
|                 56.7 |
+----------------------+
```

- Y(*p*) [1404]

  Returns the Y-coordinate value for the Point object *p* as a double-precision number.

```
mysql> SELECT Y(POINT(56.7, 53.34));
+----------------------+
| Y(POINT(56.7, 53.34)) |
+----------------------+
|                53.34 |
+----------------------+
```

16.5.2.3 LineString Functions

A LineString consists of Point values. You can extract particular points of a LineString, count the number of points that it contains, or obtain its length.

- EndPoint(*ls*) [1404]

  Returns the Point that is the endpoint of the LineString value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-------------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-------------------------------------+
| POINT(3 3)                          |
```

```
+-----------------------------------+
```

- `GLength(ls)` [1405]

  Returns as a double-precision number the length of the `LineString` value *`ls`* in its associated spatial reference.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+---------------------------+
| GLength(GeomFromText(@ls)) |
+---------------------------+
|           2.8284271247462 |
+---------------------------+
```

  `GLength()` [1405] is a nonstandard name. It corresponds to the OpenGIS `Length()` [797] function.

- `NumPoints(ls)` [1405]

  Returns the number of `Point` objects in the `LineString` value *`ls`*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-----------------------------+
| NumPoints(GeomFromText(@ls)) |
+-----------------------------+
|                           3 |
+-----------------------------+
```

- `PointN(ls,N)` [1405]

  Returns the *`N`*-th `Point` in the `Linestring` value *`ls`*. Points are numbered beginning with 1.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-----------------------------------+
| AsText(PointN(GeomFromText(@ls),2)) |
+-----------------------------------+
| POINT(2 2)                        |
+-----------------------------------+
```

- `StartPoint(ls)` [1405]

  Returns the `Point` that is the start point of the `LineString` value *`ls`*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+--------------------------------------+
| AsText(StartPoint(GeomFromText(@ls))) |
+--------------------------------------+
| POINT(1 1)                           |
+--------------------------------------+
```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- `IsRing(ls)` [1405]

  Returns 1 if the `LineString` value *`ls`* is closed (that is, its `StartPoint()` [1405] and `EndPoint()` [1404] values are the same) and is simple (does not pass through the same point more than once). Returns 0 if *`ls`* is not a ring, and –1 if it is `NULL`.

16.5.2.4 MultiLineString Functions

These functions return properties of MultiLineString values.

- GLength(*mls*) [1405]

  Returns as a double-precision number the length of the MultiLineString value *mls*. The length of *mls* is equal to the sum of the lengths of its elements.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+----------------------------+
| GLength(GeomFromText(@mls)) |
+----------------------------+
|            4.2426406871193 |
+----------------------------+
```

  GLength() [1405] is a nonstandard name. It corresponds to the OpenGIS Length() [797] function.

- IsClosed(*mls*) [1406]

  Returns 1 if the MultiLineString value *mls* is closed (that is, the StartPoint() [1405] and EndPoint() [1404] values are the same for each LineString in *mls*). Returns 0 if *mls* is not closed, and –1 if it is NULL.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+----------------------------+
| IsClosed(GeomFromText(@mls)) |
+----------------------------+
|                            0 |
+----------------------------+
```

16.5.2.5 Polygon Functions

These functions return properties of Polygon values.

- Area(*poly*) [1406]

  Returns as a double-precision number the area of the Polygon value *poly*, as measured in its spatial reference system.

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT Area(GeomFromText(@poly));
+--------------------------+
| Area(GeomFromText(@poly)) |
+--------------------------+
|                        4 |
+--------------------------+
```

- ExteriorRing(*poly*) [1406]

  Returns the exterior ring of the Polygon value *poly* as a LineString.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----------------------------------------+
```

```
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
+-------------------------------------------+
```

- `InteriorRingN(poly,N)` [1407]

  Returns the *N*-th interior ring for the `Polygon` value *poly* as a `LineString`. Rings are numbered beginning with 1.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+---------------------------------------------+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+---------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)             |
+---------------------------------------------+
```

- `NumInteriorRings(poly)` [1407]

  Returns the number of interior rings in the `Polygon` value *poly*.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+--------------------------------------+
| NumInteriorRings(GeomFromText(@poly)) |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

16.5.2.6 `MultiPolygon` Functions

These functions return properties of `MultiPolygon` values.

- `Area(mpoly)` [1406]

  Returns as a double-precision number the area of the `MultiPolygon` value *mpoly*, as measured in its spatial reference system.

```
mysql> SET @mpoly =
    -> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+---------------------------+
| Area(GeomFromText(@mpoly)) |
+---------------------------+
|                         8 |
+---------------------------+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- `Centroid(mpoly)` [1407]

  Returns the mathematical centroid for the `MultiPolygon` value *mpoly* as a `Point`. The result is not guaranteed to be on the `MultiPolygon`.

- `PointOnSurface(mpoly)`

  Returns a `Point` value that is guaranteed to be on the `MultiPolygon` value *mpoly*.

16.5.2.7 `GeometryCollection` Functions

These functions return properties of `GeometryCollection` values.

- `GeometryN(gc,N)` [1408]

  Returns the *N*-th geometry in the `GeometryCollection` value *gc*. Geometries are numbered beginning with 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+--------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+--------------------------------------+
| POINT(1 1)                           |
+--------------------------------------+
```

- `NumGeometries(gc)` [1408]

  Returns the number of geometries in the `GeometryCollection` value *gc*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+---------------------------------+
| NumGeometries(GeomFromText(@gc)) |
+---------------------------------+
|                               2 |
+---------------------------------+
```

16.5.3 Functions That Create New Geometries from Existing Ones

The following sections describe functions that take geometry values as arguments and return new geometry values.

16.5.3.1 Geometry Functions That Produce New Geometries

Section 16.5.2, “`Geometry` Functions”, discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- `Envelope(g)` [1403]
- `StartPoint(ls)` [1405]
- `EndPoint(ls)` [1404]
- `PointN(ls,N)` [1405]
- `ExteriorRing(poly)` [1406]
- `InteriorRingN(poly,N)` [1407]
- `GeometryN(gc,N)` [1408]

16.5.3.2 Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement spatial operators.

These functions are not implemented in MySQL.

- `Buffer(g,d)` [1409]

  Returns a geometry that represents all points whose distance from the geometry value *g* is less than or equal to a distance of *d*.

- `ConvexHull(g)` [1409]

  Returns a geometry that represents the convex hull of the geometry value *g*.

- `Difference(g1,g2)` [1409]

  Returns a geometry that represents the point set difference of the geometry value *g1* with *g2*.

- `Intersection(g1,g2)` [1409]

  Returns a geometry that represents the point set intersection of the geometry values *g1* with *g2*.

- `SymDifference(g1,g2)` [1409]

  Returns a geometry that represents the point set symmetric difference of the geometry value *g1* with *g2*.

- `Union(g1,g2)` [1409]

  Returns a geometry that represents the point set union of the geometry values *g1* and *g2*.

16.5.4 Functions for Testing Spatial Relations Between Geometric Objects

The functions described in these sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

16.5.5 Relations on Geometry Minimal Bounding Rectangles (MBRs)

MySQL provides several functions that test relations between minimal bounding rectangles of two geometries `g1` and `g2`. The return values 1 and 0 indicate true and false, respectively.

- `MBRContains(g1,g2)` [1409]

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* contains the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as `MBRWithin()` [1410].

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
----------------------+----------------------+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
```

- `MBRDisjoint(g1,g2)` [1409]

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- `MBREqual(g1,g2)` [1409]

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are the same.

- `MBRIntersects(g1,g2)` [1410]

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* intersect.

- `MBROverlaps(g1,g2)` [1410]

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* overlap. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `MBRTouches(g1,g2)` [1410]

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* touch. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `MBRWithin(g1,g2)` [1410]

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* is within the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as `MBRContains()` [1409].

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-------------------+-------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-------------------+-------------------+
|                 1 |                 0 |
+-------------------+-------------------+
```

16.5.6 Functions That Test Spatial Relationships Between Geometries

The OpenGIS specification defines the following functions. They test the relationship between two geometry values `g1` and `g2`.

The return values 1 and 0 indicate true and false, respectively.

Note

Currently, MySQL does not implement these functions according to the specification. Those that are implemented return the same result as the corresponding MBR-based functions. This includes functions in the following list other than `Distance()` [1411] and `Related()` [1411].

- `Contains(g1,g2)` [1410]

  Returns 1 or 0 to indicate whether *g1* completely contains *g2*. This tests the opposite relationship as `Within()` [1411].

- `Crosses(g1,g2)` [1410]

  Returns 1 if *g1* spatially crosses *g2*. Returns `NULL` if `g1` is a `Polygon` or a `MultiPolygon`, or if *g2* is a `Point` or a `MultiPoint`. Otherwise, returns 0.

  The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

  - The two geometries intersect
  - Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
  - Their intersection is not equal to either of the two given geometries

- `Disjoint(g1,g2)` [1411]

  Returns 1 or 0 to indicate whether *`g1`* is spatially disjoint from (does not intersect) *`g2`*.

- `Distance(g1,g2)` [1411]

  Returns as a double-precision number the shortest distance between any two points in the two geometries.

- `Equals(g1,g2)` [1411]

  Returns 1 or 0 to indicate whether *`g1`* is spatially equal to *`g2`*.

- `Intersects(g1,g2)` [1411]

  Returns 1 or 0 to indicate whether *`g1`* spatially intersects *`g2`*.

- `Overlaps(g1,g2)` [1411]

  Returns 1 or 0 to indicate whether *`g1`* spatially overlaps *`g2`*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `Related(g1,g2,pattern_matrix)`

  Returns 1 or 0 to indicate whether the spatial relationship specified by *`pattern_matrix`* exists between *`g1`* and *`g2`*. Returns –1 if the arguments are `NULL`. The pattern matrix is a string. Its specification will be noted here if this function is implemented.

- `Touches(g1,g2)` [1411]

  Returns 1 or 0 to indicate whether *`g1`* spatially touches *`g2`*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `Within(g1,g2)` [1411]

  Returns 1 or 0 to indicate whether *`g1`* is spatially within *`g2`*. This tests the opposite relationship as `Contains()` [1410].

16.6 Optimizing Spatial Analysis

For `MyISAM` tables, Search operations in nonspatial databases can be optimized using `SPATIAL` indexes. This is true for spatial databases as well. With the help of a great variety of multi-dimensional indexing methods that have previously been designed, it is possible to optimize spatial searches. The most typical of these are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** for SPATIAL indexes on spatial columns. A SPATIAL index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

16.6.1 Creating Spatial Indexes

For MyISAM tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but extended with the SPATIAL keyword. Currently, columns in spatial indexes must be declared NOT NULL. The following examples demonstrate how to create spatial indexes:

- With CREATE TABLE:

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g)) ENGINE=MyISAM;
```

- With ALTER TABLE:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With CREATE INDEX:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

For MyISAM tables, SPATIAL INDEX creates an R-tree index. For nonspatial indexing of spatial columns, MyISAM tables creates a B-tree index. A B-tree index on spatial values will be useful for exact-value lookups, but not for range scans.

For more information on indexing spatial columns, see Section 12.1.4, “CREATE INDEX Syntax”.

To drop spatial indexes, use ALTER TABLE or DROP INDEX:

- With ALTER TABLE:

```
ALTER TABLE geom DROP INDEX g;
```

- With DROP INDEX:

```
DROP INDEX sp_index ON geom;
```

Example: Suppose that a table geom contains more than 32,000 geometries, which are stored in the column g of type GEOMETRY. The table also has an AUTO_INCREMENT column fid for storing object ID values.

```
mysql> DESCRIBE geom;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| fid   | int(11)  |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-------+----------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+----------+
| count(*) |
+----------+
```

```
|    32376 |
+----------+
1 row in set (0.00 sec)
```

To add a spatial index on the column `g`, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

16.6.2 Using a Spatial Index

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` [1409] or `MBRWithin()` [1410] in the `WHERE` clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | AsText(g)                                                   |
+-----+-------------------------------------------------------------+
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.00 sec)
```

Use `EXPLAIN` to check the way this query is executed:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: range
possible_keys: g
          key: g
      key_len: 32
          ref: NULL
         rows: 50
        Extra: Using where
```

```
1 row in set (0.00 sec)
```

Check what would happen without a spatial index:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 32376
        Extra: Using where
1 row in set (0.00 sec)
```

Executing the `SELECT` statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | AsText(g)                                                   |
+-----+-------------------------------------------------------------+
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.46 sec)
```

16.7 MySQL Conformance and Compatibility

MySQL does not yet implement the following GIS features:

- Additional Metadata Views

  OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` [797] on `LineString` and `MultiLineString` currently should be called in MySQL as `GLength()` [1405]

  The problem is that there is an existing SQL function `Length()` [797] that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context. We need either to solve this somehow, or decide on another function name.

Chapter 17 Connectors and APIs

Table of Contents

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both Connectors and the APIs enable you to connect and execute MySQL statements from another language or environment, including ODBC, Java (JDBC), Perl, Python, PHP, Ruby, and native C and embedded MySQL instances.

Note

Connector version numbers do not correlate with MySQL Server version numbers. See Table 17.2, “MySQL Connector Versions and MySQL Server Versions”.

MySQL Connectors

Oracle develops a number of connectors:

- Connector/ODBC provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix, and Mac OS X platforms.
- Connector/Net enables developers to create .NET applications that connect to MySQL. Connector/Net implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that use Connector/Net can be written in any supported .NET language.

  The MySQL Visual Studio Plugin works with Connector/Net and Visual Studio 2005. The plugin is a MySQL DDEX Provider, which means that you can use the schema and data manipulation tools available in Visual Studio to create and edit objects within a MySQL database.
- Connector/J provides driver support for connecting to MySQL from Java applications using the standard Java Database Connectivity (JDBC) API.
- Connector/C++ enables C++ applications to connect to MySQL.
- Connector/C is a standalone replacement for the MySQL Client Library (`libmysqlclient`), to be used for C applications.

The MySQL C API

For direct access to using MySQL natively within a C application, there are two methods:

- The C API provides low-level access to the MySQL client/server protocol through the `libmysqlclient` client library. This is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command-line clients and many of the MySQL Connectors and third-party APIs detailed here.

  `libmysqlclient` is included in MySQL distributions and in MySQL Connector/C distributions.
- `libmysqld` is an embedded MySQL server library that enables you to embed an instance of the MySQL server into your C applications.

  `libmysqld` is included in MySQL distributions, but not in MySQL Connector/C distributions.

See also Section 17.6.1, “MySQL C API Implementations”.

To access MySQL from a C application, or to build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the C API is where to start. A number of programmer's utilities are available to help with the process; see Section 4.7, “MySQL Program Development Utilities”.

Third-Party MySQL APIs

The remaining APIs described in this chapter provide an interface to MySQL from specific application languages. These third-party solutions are not developed or supported by Oracle. Basic information on their usage and abilities is provided here for reference purposes only.

All the third-party language APIs are developed using one of two methods, using `libmysqlclient` or by implementing a *native driver*. The two solutions offer different benefits:

- Using *`libmysqlclient`* offers complete compatibility with MySQL because it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and

interfaces exposed through `libmysqlclient` and the performance may be lower as data is copied between the native language, and the MySQL API components.

- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier for end users to build and deploy because no copy of the MySQL client libraries is needed to build the native driver components.

Table 17.1, “MySQL APIs and Interfaces” lists many of the libraries and interfaces available for MySQL are shown in Table 17.1, “MySQL APIs and Interfaces” Table 17.2, “MySQL Connector Versions and MySQL Server Versions” shows which MySQL Server versions each connector supports.

Table 17.1 MySQL APIs and Interfaces

Environment	API	Type	Notes
Ada	GNU Ada MySQL Bindings	`libmysqlclient`	See MySQL Bindings for GNU Ada
C	C API	`libmysqlclient`	See Section 17.6, “MySQL C API”.
C	Connector/C	Replacement for `libmysqlclient`	See MySQL Connector/C Developer Guide.
C++	Connector/C++	`libmysqlclient`	See MySQL Connector/C++ Developer Guide.
	MySQL++	`libmysqlclient`	See MySQL++ Web site.
	MySQL wrapped	`libmysqlclient`	See MySQL wrapped.
Cocoa	MySQL-Cocoa	`libmysqlclient`	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	`libmysqlclient`	See MySQL for D.
Eiffel	Eiffel MySQL	`libmysqlclient`	See Section 17.12, “MySQL Eiffel Wrapper”.
Erlang	`erlang-mysql-driver`	`libmysqlclient`	See `erlang-mysql-driver`.
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O'Sullivan's pure Haskell MySQL bindings.
	`hsql-mysql`	`libmysqlclient`	See MySQL driver for Haskell .
Java/ JDBC	Connector/J	Native Driver	See MySQL Connector/J Developer Guide.
Kaya	MyDB	`libmysqlclient`	See MyDB.
Lua	LuaSQL	`libmysqlclient`	See LuaSQL.
.NET/ Mono	Connector/Net	Native Driver	See MySQL Connector/Net Developer Guide.
Objective Caml	OBjective Caml MySQL Bindings	`libmysqlclient`	See MySQL Bindings for Objective Caml.
Octave	Database bindings for GNU Octave	`libmysqlclient`	See Database bindings for GNU Octave.
ODBC	Connector/ODBC	`libmysqlclient`	See MySQL Connector/ODBC Developer Guide.
Perl	`DBI`/`DBD::mysql`	`libmysqlclient`	See Section 17.8, “MySQL Perl API”.

Environment	API	Type	Notes
	`Net::MySQL`	Native Driver	See `Net::MySQL` at CPAN
PHP	`mysql`, `ext/mysql` interface (deprecated)	`libmysqlclient`	See Original MySQL API (`Mysql`).
	`mysqli`, `ext/mysqli` interface	`libmysqlclient`	See MySQL Improved Extension (`Mysqli`).
	`PDO_MYSQL`	`libmysqlclient`	See MySQL Functions (PDO_MYSQL) (`MySQL (PDO)`).
	PDO mysqlnd	Native Driver	
Python	MySQLdb	`libmysqlclient`	See Section 17.9, “MySQL Python API”.
Ruby	MySQL/Ruby	`libmysqlclient`	Uses `libmysqlclient`. See Section 17.10.1, “The MySQL/Ruby API”.
	Ruby/MySQL	Native Driver	See Section 17.10.2, “The Ruby/MySQL API”.
Scheme	`Myscsh`	`libmysqlclient`	See `Myscsh`.
SPL	`sql_mysql`	`libmysqlclient`	See `sql_mysql` for SPL.
Tcl	MySQLtcl	`libmysqlclient`	See Section 17.11, “MySQL Tcl API”.

Table 17.2 MySQL Connector Versions and MySQL Server Versions

Connector	Connector version	MySQL Server version
Connector/C++	1.0.5 GA	5.1, 5.4
Connector/J	5.1.8	4.1, 5.0, 5.1, 5.4
Connector/Net	1.0 (No longer supported)	4.0, 5.0
Connector/Net	5.2	5.0, 5.1, 5.4
Connector/Net	6.0	5.0, 5.1, 5.4
Connector/Net	6.1	5.0, 5.1, 5.4
Connector/ODBC	3.51 (Unicode not supported)	4.1, 5.0, 5.1, 5.4
Connector/ODBC	5.1	4.1.1+, 5.0, 5.1, 5.4

17.1 MySQL Connector/ODBC

The MySQL Connector/ODBC manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/ODBC Developer Guide
- Release notes: MySQL Connector/ODBC Release Notes

17.2 MySQL Connector/Net

The MySQL Connector/Net manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/Net Developer Guide
- Release notes: MySQL Connector/Net Release Notes

17.3 MySQL Connector/J

The MySQL Connector/J manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/J Developer Guide
- Release notes: MySQL Connector/J Release Notes

17.4 MySQL Connector/C

The MySQL Connector/C manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C Developer Guide
- Release notes: MySQL Connector/C Release Notes

17.5 libmysqld, the Embedded MySQL Server Library

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change an old threaded application to use the embedded library, you normally only have to add calls to the following functions.

Function	When to Call
`mysql_library_init()`	Should be called before any other MySQL function is called, preferably early in the `main()` function.
`mysql_library_end()`	Should be called before your program exits.
`mysql_thread_init()`	Should be called in each thread you create that accesses MySQL.
`mysql_thread_end()`	Should be called before calling `pthread_exit()`

Then you must link your code with `libmysqld.a` instead of `libmysqlclient.a`. To ensure binary compatibility between your application and the server library, be sure to compile your application against headers for the same series of MySQL that was used to compile the server library. For example, if `libmysqld` was compiled against MySQL 4.1 headers, do not compile your application against MySQL 5.1 headers, or vice versa.

The `mysql_library_`*`xxx`*`()` functions are also included in `libmysqlclient.a` to enable you to change between the embedded and the client/server version by just linking your application with the right library. See Section 17.6.6.38, “mysql_library_init()”.

One difference between the embedded server and the standalone server is that for the embedded server, authentication for connections is disabled by default. To use authentication for the embedded server, specify the `--with-embedded-privilege-control` option when you invoke `configure` to configure your MySQL distribution. This option is available as of MySQL 4.1.3.

17.5.1 Compiling Programs with libmysqld

In precompiled binary MySQL distributions that include libmysqld, the embedded server library, MySQL builds the library using the appropriate vendor compiler if there is one.

To get a libmysqld library if you build MySQL from source yourself, you should configure MySQL with the --with-embedded-server [95] option. See Section 2.9.3, “MySQL Source-Configuration Options”.

When you link your program with libmysqld, you must also include the system-specific pthread libraries and some libraries that the MySQL server uses. You can get the full list of libraries by executing mysql_config --libmysqld-libs.

The correct flags for compiling and linking a threaded program must be used, even if you do not directly call any thread functions in your code.

To compile a C program to include the necessary files to embed the MySQL server library into an executable version of a program, the compiler will need to know where to find various files and need instructions on how to compile the program. The following example shows how a program could be compiled from the command line, assuming that you are using gcc, use the GNU C compiler:

```
gcc mysql_test.c -o mysql_test -lz \
`/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Immediately following the gcc command is the name of the C program source file. After it, the -o option is given to indicate that the file name that follows is the name that the compiler is to give to the output file, the compiled program. The next line of code tells the compiler to obtain the location of the include files and libraries and other settings for the system on which it is compiled. Because of a problem with mysql_config, the option -lz (for compression) is added here. The mysql_config command is contained in backticks, not single quotation marks.

On some non-gcc platforms, the embedded library depends on C++ runtime libraries and linking against the embedded library might result in missing-symbol errors. To solve this, link using a C++ compiler or explicitly list the required libraries on the link command line.

17.5.2 Restrictions When Using the Embedded MySQL Server

The embedded server has the following limitations:

- No support for ISAM tables. (This is done mainly to make the library smaller.)
- No user-defined functions (UDFs).
- No stack trace on core dump.
- No internal RAID support. (This is not normally needed as most current operating systems support big files.)
- You cannot set this up as a master or a slave (no replication).
- Very large result sets may be unusable on low memory systems.
- You cannot connect to an embedded server from an outside process with sockets or TCP/IP. However, you can connect to an intermediate application, which in turn can connect to an embedded server on the behalf of a remote client or outside process.
- InnoDB is not reentrant in the embedded server and cannot be used for multiple connections, either successively or simultaneously.

Some of these limitations can be changed by editing the `mysql_embed.h` include file and recompiling MySQL.

17.5.3 Options with the Embedded Server

Any options that may be given with the `mysqld` server daemon, may be used with an embedded server library. Server options may be given in an array as an argument to the `mysql_library_init()`, which initializes the server. They also may be given in an option file like `my.cnf`. To specify an option file for a C program, use the `--defaults-file` [235] option as one of the elements of the second argument of the `mysql_library_init()` function. See Section 17.6.6.38, “`mysql_library_init()`”, for more information on the `mysql_library_init()` function.

Using option files can make it easier to switch between a client/server application and one where MySQL is embedded. Put common options under the `[server]` group. These are read by both MySQL versions. Client/server-specific options should go under the `[mysqld]` section. Put options specific to the embedded MySQL server library in the `[embedded]` section. Options specific to applications go under section labeled `[ApplicationName_SERVER]`. See Section 4.2.3.3, “Using Option Files”.

17.5.4 Embedded Server Examples

These two example programs should work without any changes on a Linux or FreeBSD system. For other operating systems, minor changes are needed, mostly with file paths. These examples are designed to give enough details for you to understand the problem, without the clutter that is a necessary part of a real application. The first example is very straightforward. The second example is a little more advanced with some error checking. The first is followed by a command-line entry for compiling the program. The second is followed by a GNUmake file that may be used for compiling instead.

Example 1

`test1_libmysqld.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = { "mysql_test", "--defaults-file=my.cnf", NULL };
int num_elements = (sizeof(server_options) / sizeof(char *)) - 1;

static char *server_groups[] = { "libmysqld_server", "libmysqld_client", NULL };

int main(void)
{
   mysql_library_init(num_elements, server_options, server_groups);
   mysql = mysql_init(NULL);
   mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
   mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

   mysql_real_connect(mysql, NULL,NULL,NULL, "database1", 0,NULL,0);

   mysql_query(mysql, "SELECT column1, column2 FROM table1");

   results = mysql_store_result(mysql);

   while((record = mysql_fetch_row(results))) {
      printf("%s - %s \n", record[0], record[1]);
   }
```

```
   mysql_free_result(results);
   mysql_close(mysql);
   mysql_library_end();

   return 0;
}
```

Here is the command line for compiling the above program:

```
gcc test1_libmysqld.c -o test1_libmysqld -lz \
 `/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Example 2

To try the example, create a `test2_libmysqld` directory at the same level as the MySQL source directory. Save the `test2_libmysqld.c` source and the `GNUmakefile` in the directory, and run GNU `make` from inside the `test2_libmysqld` directory.

`test2_libmysqld.c`

```
/*
 * A simple example client, using the embedded MySQL server library
*/

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
  "test2_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
  MYSQL *one, *two;

  /* mysql_library_init() must be called before any other mysql
   * functions.
   *
   * You can use mysql_library_init(0, NULL, NULL), and it
   * initializes the server using groups = {
   *   "server", "embedded", NULL
   *  }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test2_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

   * You could, of course, modify argc and argv before passing
   * them to this function.  Or you could create new ones in any
   * way you like.  But all of the arguments in argv (except for
   * argv[0], which is the program name) should be valid options
   * for the MySQL server.
   *
   * If you link this client against the normal mysqlclient
   * library, this function is just a stub that does nothing.
```

```
   */
  mysql_library_init(argc, argv, (char **)server_groups);

  one = db_connect("test");
  two = db_connect(NULL);

  db_do_query(one, "SHOW TABLE STATUS");
  db_do_query(two, "SHOW DATABASES");

  mysql_close(two);
  mysql_close(one);

  /* This must be called after all other mysql functions */
  mysql_library_end();

  exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
  va_list ap;
  va_start(ap, fmt);
  vfprintf(stderr, fmt, ap);
  va_end(ap);
  (void)putc('\n', stderr);
  if (db)
    db_disconnect(db);
  exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
  MYSQL *db = mysql_init(NULL);
  if (!db)
    die(db, "mysql_init failed: no memory");
  /*
   * Notice that the client and server use separate group names.
   * This is critical, because the server does not accept the
   * client's options, and vice versa.
   */
  mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
  if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
    die(db, "mysql_real_connect failed: %s", mysql_error(db));

  return db;
}

void
db_disconnect(MYSQL *db)
{
  mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
  if (mysql_query(db, query) != 0)
    goto err;

  if (mysql_field_count(db) > 0)
  {
    MYSQL_RES   *res;
    MYSQL_ROW    row, end_row;
    int num_fields;
```

```
    if (!(res = mysql_store_result(db)))
      goto err;
    num_fields = mysql_num_fields(res);
    while ((row = mysql_fetch_row(res)))
    {
      (void)fputs(">> ", stdout);
      for (end_row = row + num_fields; row < end_row; ++row)
        (void)printf("%s\t", row ? (char*)*row : "NULL");
      (void)fputc('\n', stdout);
    }
    (void)fputc('\n', stdout);
    mysql_free_result(res);
  }
  else
    (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

  return;

err:
  die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}
```

GNUmakefile

```
# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc      := $(HOME)/mysql-4.0/include
#lib      := $(HOME)/mysql-4.0/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS    = -L$(lib) -lmysqld -lz -lm -ldl -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
        rm -f $(targets) $(objects) *.core
```

17.5.5 Licensing the Embedded Server

We encourage everyone to promote free software by releasing code under the GPL or a compatible license. For those who are not able to do this, another option is to purchase a commercial license for the MySQL code from Oracle Corporation. For details, please see http://www.mysql.com/company/legal/licensing/.

17.6 MySQL C API

The C API provides low-level access to the MySQL client/server protocol and enables C programs to access database contents. The C API code is distributed with MySQL and implemented in the `libmysqlclient` library See Section 17.6.1, “MySQL C API Implementations”.

Most other client APIs use the `libmysqlclient` library to communicate with the MySQL server. (Exceptions are except Connector/J and Connector/Net.) This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs because they are referenced from the library. For a list of these variables, see Section 4.1, “Overview of MySQL Programs”.

For instructions on building client programs using the C API, see Section 17.6.3.1, “Building C API Client Programs”. For programming with threads, see Section 17.6.3.2, “Writing C API Threaded Client Programs”. To create a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see Section 17.5, “libmysqld, the Embedded MySQL Server Library”.

Note

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, the programs were probably compiled using old header or library files. In this case, check the date of the `mysql.h` file and `libmysqlclient.a` library used for compilation to verify that they are from the new MySQL distribution. If not, recompile he programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.15` to `libmysqlclient.so.16`).

Clients have a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (16MB by default). Because buffer sizes are increased only as demand warrants, simply increasing the maximum limit does not in itself cause more resources to be used. This size check is mostly a precaution against erroneous statements and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each session's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The default maximum built into the client library is 1GB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` [418] parameter at server startup. See Section 7.8.2, “Tuning Server Parameters”.

The MySQL server shrinks each communication buffer to `net_buffer_length` [422] bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

17.6.1 MySQL C API Implementations

The MySQL C API is a C-based API that client applications written in C can use to communicate with MySQL Server. Client programs refer to C API header files at compile time and link to a C API library file at link time. The library comes in two versions, depending on how the application is intended to communicate with the server:

- `libmysqlclient`: The client version of the library, used for applications that communicate over a network connection as a client of a standalone server process.

- `libmysqld`: The embedded server version of the library, used for applications intended to include an embedded MySQL server within the application itself. The application communicates with its own private server instance.

Both libraries have the same interface. In terms of C API calls, an application communicates with a standalone server the same way it communicates with an embedded server. A given client can be built to communicate with a standalone or embedded server, depending on whether it is linked against `libmysqlclient` or `libmysqld` at build time.

There are two ways to obtain the C API header and library files required to build C API client programs:

- Install a MySQL Server distribution. Server distributions include both `libmysqlclient` and `libmysqld`.
- Install a MySQL Connector/C distribution. Connector/C distributions include only `libmysqlclient`. They do not include `libmysqld`.

For both MySQL Server and MySQL Connector/C, you can install a binary distribution that contains the C API files pre-built, or you can use a source distribution and build the C API files yourself.

Normally, you install either a MySQL Server distribution or a MySQL Connector/C distribution, but not both. It is possible that installing both can cause problems.

17.6.2 Example C API Client Programs

Many of the clients in MySQL source distributions are written in C, such as `mysql`, `mysqladmin`, and `mysqlshow`. If you are looking for examples that demonstrate how to use the C API, take a look at these clients: Obtain a source distribution and look in its `client` directory. See Section 2.1.3, “How to Get MySQL”.

17.6.3 Building and Running C API Client Programs

The following sections provide information on building client programs that use the C API. Topics include compiling and linking clients, writing threaded clients, and troubleshooting runtime problems.

17.6.3.1 Building C API Client Programs

This section provides guidelines for compiling C programs that use the MySQL C API.

Compiling MySQL Clients on Unix

You may need to specify an `-I` option when you compile client programs that use MySQL header files, so that the compiler can find them. For example, if the header files are installed in `/usr/local/mysql/include`, use this option in the compile command:

```
-I/usr/local/mysql/include
```

MySQL clients must be linked using the `-lmysqlclient -lz` options in the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use these options in the link command:

```
-L/usr/local/mysql/lib -lmysqlclient -lz
```

The path names may differ on your system. Adjust the `-I` and `-L` options as necessary.

To make it simpler to compile MySQL programs on Unix, use the `mysql_config` script. See Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”.

`mysql_config` displays the options needed for compiling or linking:

```
shell> mysql_config --cflags
shell> mysql_config --libs
```

You can run those commands to get the proper options and add them manually to compilation or link commands. Alternatively, include the output from `mysql_config` directly within command lines using backticks:

```
shell> gcc -c `mysql_config --cflags` progname.c
shell> gcc -o progname progname.o `mysql_config --libs`
```

Compiling MySQL Clients on Microsoft Windows

To specify header and library file locations, use the facilities provided by your development environment.

On Windows, in your source files, you should include `my_global.h` before `mysql.h`:

```
#include <my_global.h>
#include <mysql.h>
```

`my_global.h` includes any other files needed for Windows compatibility (such as `windows.h`) if you compile your program on Windows.

You can either link your code with the dynamic `libmysql.lib` library, which is just a wrapper to load in `libmysql.dll` on demand, or link with the static `mysqlclient.lib` library.

The MySQL client libraries are compiled as threaded libraries, so you should also compile your code to be multi-threaded.

Troubleshooting Problems Linking to the MySQL Client Library

Linking with the single-threaded library (`libmysqlclient`) may lead to linker errors related to `pthread` symbols. When using the single-threaded library, please compile your client with `MYSQL_CLIENT_NO_THREADS` defined. This can be done on the command line by using the `-D` option to the compiler, or in your source code before including the MySQL header files. This define should not be used when building for use with the thread-safe client library (`libmysqlclient_r`).

If the linker cannot find the MySQL client library, you might get undefined-reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-L`*`dir_path`* `-lmysqlclient` at the end of your link command, where *`dir_path`* represents the path name of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from `mysql_config` might indicate other libraries that should be specified on the link command as well. You can include `mysql_config` output directly in your compile or link command using backticks. For example:

```
shell> gcc -o progname progname.o `mysql_config --libs`
```

If an error occurs at link time that the `floor` symbol is undefined, link to the math library by adding `-lm` to the end of the compile/link line.

If you get `undefined reference` errors for the `uncompress` or `compress` function, add `-lz` to the end of your link command and try again.

Similarly, if you get undefined-reference errors for other functions that should exist on your system, such as `connect()`, check the manual page for the function in question to determine which libraries you should add to the link command.

If you get undefined-reference errors such as the following for functions that do not exist on your system, it usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

In this case, you should download the latest MySQL or MySQL Connector/C source distribution and compile the MySQL client library yourself. See Section 2.9, “Installing MySQL from Source”, and MySQL Connector/C Developer Guide.

17.6.3.2 Writing C API Threaded Client Programs

The client library is almost thread-safe. The biggest problem is that the subroutines in `net.c` that read from sockets are not interrupt-safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, socket handling should be thread-safe.

Note

Beginning with version 4.0.6, MySQL blocks `SIGPIPE` on the first call to `mysql_server_init()`, `mysql_init()`, or `mysql_connect()`. This is done to avoid aborting the program when a connection terminates. To use your own `SIGPIPE` handler, first call `mysql_server_init()`, then install your handler. As of MySQL 4.1.10, use `mysql_library_init()` instead of `mysql_server_init()`.

Before MySQL 4.0, binary client libraries that we provided other than those for Windows were not normally compiled with the thread-safe option. Current binary distributions should have both a normal client library, `libmysqlclient`, and a thread-safe library, `libmysqlclient_r`. For threaded clients, link against the latter library. If “undefined symbol” errors occur, in most cases this is because you have not included the thread libraries on the link/compile command.

The thread-safe client library, `libmysqlclient_r`, is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Multiple threads cannot send a query to the MySQL server at the same time on the same connection. In particular, you must ensure that between calls to `mysql_query()` and `mysql_store_result()` in one thread, no other thread uses the same connection. You must have a mutex lock around your pair of `mysql_query()` and `mysql_store_result()` calls. After `mysql_store_result()` returns, the lock can be released and other threads may query the same connection.

  If you use POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

- Many threads can access different result sets that are retrieved with `mysql_store_result()`.
- To use `mysql_use_result()`, you must ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use `mysql_store_result()`.
- `mysql_ping()` does not attempt a reconnection if the connection is down. It returns an error instead.

You need to know the following if you have a thread that did not create the connection to the MySQL database but is calling MySQL functions:

When you call `mysql_init()`, MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things). If you call a MySQL function before the thread has called `mysql_init()`, the thread does not have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later. To avoid problems, you must do the following:

1. Call `mysql_library_init()` before any other MySQL functions. It is not thread-safe, so call it before threads are created, or protect the call with a mutex.
2. Arrange for `mysql_thread_init()` to be called early in the thread handler before calling any MySQL function. If you call `mysql_init()`, it will call `mysql_thread_init()` for you.
3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

The preceding notes regarding `mysql_init()` also apply to `mysql_connect()`, which calls `mysql_init()`.

17.6.3.3 Running C API Client Programs

Undefined-reference errors might occur at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `libmysqlclient` library cannot be found, it means that your system cannot find the shared `libmysqlclient.so` library. The solution to this problem is to tell your system to search for shared libraries in the directory where that library is located. Use whichever of the following methods is appropriate for your system:

- Add the path of the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` or `LD_LIBRARY` environment variable.
- On Mac OS X, add the path of the directory where `libmysqlclient.dylib` is located to the `DYLD_LIBRARY_PATH` environment variable.
- Copy the shared-library files (such as `libmysqlclient.so`) to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`. Be sure to copy all related files. A shared library might exist under several names, using symlinks to provide the alternate names.

Another way to solve this problem is by linking your program statically with the `-static` option, or by removing the dynamic MySQL libraries before linking your code. Before trying the second method, you should be sure that no other programs are using the dynamic libraries.

17.6.4 C API Data Structures

This section describes C API data structures other than those used for prepared statements. For information about the latter, see Section 17.6.8, “C API Prepared Statement Data Structures”.

- `MYSQL`

This structure represents a handle to one database connection. It is used for almost all MySQL functions. You should not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy will be usable.

- `MYSQL_RES`

  This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

  This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

- `MYSQL_FIELD`

  This structure contains metadata: information about a field, such as the field's name, type, and size. Its members are described in more detail later in this section. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

- `MYSQL_FIELD_OFFSET`

  This is a type-safe representation of an offset into a MySQL field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

- `my_ulonglong`

  The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()`, and `mysql_insert_id()`. This type provides a range of `0` to `1.84e19`.

  On some systems, attempting to print a value of type `my_ulonglong` does not work. To print such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

```
printf ("Number of rows: %lu\n",
        (unsigned long) mysql_num_rows(result));
```

- `my_bool`

  A boolean type, for values that are true (nonzero) or false (zero).

The `MYSQL_FIELD` structure contains the members described in the following list:

- `char * name`

  The name of the field, as a null-terminated string. If the field was given an alias with an `AS` clause, the value of `name` is the alias.

- `char * org_name`

  The name of the field, as a null-terminated string. Aliases are ignored. For expressions, the value is an empty string. This member was added in MySQL 4.1.0.

- `char * table`

The name of the table containing this field, if it is not a calculated field. For calculated fields, the `table` value is an empty string. If the table was given an alias with an `AS` clause, the value of `table` is the alias. For a `UNION`, the value is the empty string as of MySQL 4.0.26.

- `char * org_table`

  The name of the table, as a null-terminated string. Aliases are ignored. For a `UNION`, the value is the empty string as of MySQL 4.0.26. This member was added in MySQL 4.1.0.

- `char * db`

  The name of the database that the field comes from, as a null-terminated string. If the field is a calculated field, `db` is an empty string. For a `UNION`, the value is the empty string as of MySQL 4.0.26. This member was added in MySQL 4.1.0.

- `char * catalog`

  The catalog name. This value is always `"def"`. This member was added in MySQL 4.1.1.

- `char * def`

  The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `unsigned long length`

  The width of the field. This corresponds to the display length, in bytes.

  The server determines the `length` value before it generates the result set, so this is the minimum length required for a data type capable of holding the largest possible value from the result column, without knowing in advance the actual values that will be produced by the query for the result set.

- `unsigned long max_length`

  The maximum width of the field for the result set (the length in bytes of the longest field value for the rows actually in the result set). If you use `mysql_store_result()` or `mysql_list_fields()`, this contains the maximum length for the field. If you use `mysql_use_result()`, the value of this variable is zero.

  The value of `max_length` is the length of the string representation of the values in the result set. For example, if you retrieve a `FLOAT` column and the “widest” value is `-12.345`, `max_length` is 7 (the length of `'-12.345'`).

  If you are using prepared statements, `max_length` is not set by default because for the binary protocol the lengths of the values depend on the types of the values in the result set. (See Section 17.6.8, “C API Prepared Statement Data Structures”.) If you want the `max_length` values anyway, enable the `STMT_ATTR_UPDATE_MAX_LENGTH` option with `mysql_stmt_attr_set()` and the lengths will be set when you call `mysql_stmt_store_result()`. (See Section 17.6.10.3, “`mysql_stmt_attr_set()`”, and Section 17.6.10.27, “`mysql_stmt_store_result()`”.)

- `unsigned int name_length`

  The length of `name`. This member was added in MySQL 4.1.0.

- `unsigned int org_name_length`

  The length of `org_name`. This member was added in MySQL 4.1.0.

- `unsigned int table_length`

  The length of `table`. This member was added in MySQL 4.1.0.

- `unsigned int org_table_length`

  The length of `org_table`. This member was added in MySQL 4.1.0.

- `unsigned int db_length`

  The length of `db`. This member was added in MySQL 4.1.0.

- `unsigned int catalog_length`

  The length of `catalog`. This member was added in MySQL 4.1.1.

- `unsigned int def_length`

  The length of `def`. This member was added in MySQL 4.1.0.

- `unsigned int flags`

  Bit-flags that describe the field. The `flags` value may have zero or more of the following bits set.

Flag Value	Flag Description
`NOT_NULL_FLAG`	Field cannot be `NULL`
`PRI_KEY_FLAG`	Field is part of a primary key
`UNIQUE_KEY_FLAG`	Field is part of a unique key
`MULTIPLE_KEY_FLAG`	Field is part of a nonunique key
`UNSIGNED_FLAG`	Field has the `UNSIGNED` attribute
`ZEROFILL_FLAG`	Field has the `ZEROFILL` attribute
`BINARY_FLAG`	Field has the `BINARY` attribute
`AUTO_INCREMENT_FLAG`	Field has the `AUTO_INCREMENT` attribute
`NUM_FLAG`	Field is numeric
`ENUM_FLAG`	Field is an `ENUM`
`SET_FLAG`	Field is a `SET`
`BLOB_FLAG`	Field is a `BLOB` or `TEXT` (deprecated)
`TIMESTAMP_FLAG`	Field is a `TIMESTAMP` (deprecated)

  Use of the `BLOB_FLAG`, `ENUM_FLAG`, `SET_FLAG`, and `TIMESTAMP_FLAG` flags is deprecated because they indicate the type of a field rather than an attribute of its type. It is preferable to test `field->type` against `MYSQL_TYPE_BLOB`, `MYSQL_TYPE_ENUM`, `MYSQL_TYPE_SET`, or `MYSQL_TYPE_TIMESTAMP` instead.

  `NUM_FLAG` indicates that a column is numeric. This includes columns with a type of `MYSQL_TYPE_DECIMAL`, `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, `MYSQL_TYPE_FLOAT`, `MYSQL_TYPE_DOUBLE`, `MYSQL_TYPE_NULL`, `MYSQL_TYPE_TIMESTAMP` (before MySQL 4.1), `MYSQL_TYPE_LONGLONG`, `MYSQL_TYPE_INT24`, and `MYSQL_TYPE_YEAR`.

  The following example illustrates a typical use of the `flags` value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field cannot be null\n");
```

You may use the following convenience macros to determine the boolean status of the `flags` value.

Flag Status	Description
`IS_NOT_NULL(flags)`	True if this field is defined as `NOT NULL`
`IS_PRI_KEY(flags)`	True if this field is a primary key
`IS_BLOB(flags)`	True if this field is a `BLOB` or `TEXT` (deprecated; test `field->type` instead)

- `unsigned int decimals`

  The number of decimals for numeric fields.

- `unsigned int charsetnr`

  An ID number that indicates the character set/collation pair for the field. This member was added in MySQL 4.1.0.

  To distinguish between binary and nonbinary data for string data types, check whether the `charsetnr` value is 63. If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

  `charsetnr` values are the same as those displayed in the `Id` column of the `SHOW COLLATION` statement. You can use this statement to see which character set and collation specific `charsetnr` values indicate:

```
mysql> SHOW COLLATION;
+----------------------+----------+-----+---------+----------+---------+
| Collation            | Charset  | Id  | Default | Compiled | Sortlen |
+----------------------+----------+-----+---------+----------+---------+
...
| utf8_general_ci      | utf8     |  33 | Yes     | Yes      |       1 |
...
| binary               | binary   |  63 | Yes     | Yes      |       1 |
...
+----------------------+----------+-----+---------+----------+---------+
```

- `enum enum_field_types type`

  The type of the field. The `type` value may be one of the `MYSQL_TYPE_` symbols shown in the following table. Before MySQL 4.1, the symbol names begin with `FIELD_TYPE_` rather than `MYSQL_TYPE_`. The older types still are recognized for backward compatibility.

Type Value	Type Description
`MYSQL_TYPE_TINY`	`TINYINT` field
`MYSQL_TYPE_SHORT`	`SMALLINT` field
`MYSQL_TYPE_LONG`	`INTEGER` field
`MYSQL_TYPE_INT24`	`MEDIUMINT` field
`MYSQL_TYPE_LONGLONG`	`BIGINT` field
`MYSQL_TYPE_DECIMAL`	`DECIMAL` or `NUMERIC` field

Type Value	Type Description
MYSQL_TYPE_FLOAT	FLOAT field
MYSQL_TYPE_DOUBLE	DOUBLE or REAL field
MYSQL_TYPE_TIMESTAMP	TIMESTAMP field
MYSQL_TYPE_DATE	DATE field
MYSQL_TYPE_TIME	TIME field
MYSQL_TYPE_DATETIME	DATETIME field
MYSQL_TYPE_YEAR	YEAR field
MYSQL_TYPE_STRING	CHAR or BINARY field
MYSQL_TYPE_VAR_STRING	VARCHAR or VARBINARY field
MYSQL_TYPE_BLOB	BLOB or TEXT field (use max_length to determine the maximum length)
MYSQL_TYPE_SET	SET field
MYSQL_TYPE_ENUM	ENUM field
MYSQL_TYPE_GEOMETRY	Spatial field (MySQL 4.1.0 and up)
MYSQL_TYPE_NULL	NULL-type field

You can use the IS_NUM() macro to test whether a field has a numeric type. Pass the type value to IS_NUM() and it evaluates to TRUE if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

17.6.5 C API Function Overview

The functions available in the C API are summarized here and described in greater detail in a later section. See Section 17.6.6, “C API Function Descriptions”.

Table 17.3 C API Function Names and Descriptions

Function	Description
my_init()	Initialize global variables, and thread handler in thread-safe programs
mysql_affected_rows()	Returns the number of rows changed/deleted/inserted by the last UPDATE, DELETE, or INSERT query
mysql_autocommit()	Toggles autocommit mode on/off
mysql_change_user()	Changes user and database on an open connection
mysql_character_set_name()	Return default character set name for current connection
mysql_close()	Closes a server connection
mysql_commit()	Commits the transaction
mysql_connect()	Connects to a MySQL server (this function is deprecated; use mysql_real_connect() instead)
mysql_create_db()	Creates a database (this function is deprecated; use the SQL statement CREATE DATABASE instead)
mysql_data_seek()	Seeks to an arbitrary row number in a query result set
mysql_debug()	Does a DBUG_PUSH with the given string

Function	Description
mysql_drop_db()	Drops a database (this function is deprecated; use the SQL statement DROP DATABASE instead)
mysql_dump_debug_info()	Makes the server write debug information to the log
mysql_eof()	Determines whether the last row of a result set has been read (this function is deprecated; mysql_errno() or mysql_error() may be used instead)
mysql_errno()	Returns the error number for the most recently invoked MySQL function
mysql_error()	Returns the error message for the most recently invoked MySQL function
mysql_escape_string()	Escapes special characters in a string for use in an SQL statement
mysql_fetch_field()	Returns the type of the next table field
mysql_fetch_field_direct()	Returns the type of a table field, given a field number
mysql_fetch_fields()	Returns an array of all field structures
mysql_fetch_lengths()	Returns the lengths of all columns in the current row
mysql_fetch_row()	Fetches the next row from the result set
mysql_field_count()	Returns the number of result columns for the most recent statement
mysql_field_seek()	Puts the column cursor on a specified column
mysql_field_tell()	Returns the position of the field cursor used for the last mysql_fetch_field()
mysql_free_result()	Frees memory used by a result set
mysql_get_client_info()	Returns client version information as a string
mysql_get_client_version()	Returns client version information as an integer
mysql_get_host_info()	Returns a string describing the connection
mysql_get_proto_info()	Returns the protocol version used by the connection
mysql_get_server_info()	Returns the server version number
mysql_get_server_version()	Returns version number of server as an integer
mysql_hex_string()	Encode string in hexadecimal format
mysql_info()	Returns information about the most recently executed query
mysql_init()	Gets or initializes a MYSQL structure
mysql_insert_id()	Returns the ID generated for an AUTO_INCREMENT column by the previous query
mysql_kill()	Kills a given thread
mysql_library_end()	Finalize the MySQL C API library
mysql_library_init()	Initialize the MySQL C API library
mysql_list_dbs()	Returns database names matching a simple regular expression
mysql_list_fields()	Returns field names matching a simple regular expression
mysql_list_processes()	Returns a list of the current server threads
mysql_list_tables()	Returns table names matching a simple regular expression
mysql_more_results()	Checks whether any more results exist

Function	Description
mysql_next_result()	Returns/initiates the next result in multiple-result executions
mysql_num_fields()	Returns the number of columns in a result set
mysql_num_rows()	Returns the number of rows in a result set
mysql_options()	Sets connect options for mysql_real_connect()
mysql_ping()	Checks whether the connection to the server is working, reconnecting as necessary
mysql_query()	Executes an SQL query specified as a null-terminated string
mysql_real_connect()	Connects to a MySQL server
mysql_real_escape_string()	Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection
mysql_real_query()	Executes an SQL query specified as a counted string
mysql_refresh()	Flush or reset tables and caches
mysql_reload()	Tells the server to reload the grant tables
mysql_rollback()	Rolls back the transaction
mysql_row_seek()	Seeks to a row offset in a result set, using value returned from mysql_row_tell()
mysql_row_tell()	Returns the row cursor position
mysql_select_db()	Selects a database
mysql_server_end()	Finalize the MySQL C API library
mysql_server_init()	Initialize the MySQL C API library
mysql_set_character_set()	Set default character set for current connection
mysql_set_local_infile_default()	Set the LOAD DATA LOCAL INFILE handler callbacks to their default values
mysql_set_local_infile_handler()	Install application-specific LOAD DATA LOCAL INFILE handler callbacks
mysql_set_server_option()	Sets an option for the connection (like multi-statements)
mysql_sqlstate()	Returns the SQLSTATE error code for the last error
mysql_shutdown()	Shuts down the database server
mysql_ssl_set()	Prepare to establish SSL connection to server
mysql_stat()	Returns the server status as a string
mysql_store_result()	Retrieves a complete result set to the client
mysql_thread_end()	Finalize thread handler
mysql_thread_id()	Returns the current thread ID
mysql_thread_init()	Initialize thread handler
mysql_thread_safe()	Returns 1 if the clients are compiled as thread-safe
mysql_use_result()	Initiates a row-by-row result set retrieval
mysql_warning_count()	Returns the warning count for the previous SQL statement

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling `mysql_library_init()`. This function exists in both the `libmysqlclient` C client library and the `libmysqld` embedded server library, so it is used whether you build a regular client program by linking with the `-libmysqlclient` flag, or an embedded server application by linking with the `-libmysqld` flag.

2. Initialize a connection handler by calling `mysql_init()` and connect to the server by calling `mysql_real_connect()`.

3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)

4. Close the connection to the MySQL server by calling `mysql_close()`.

5. End use of the MySQL library by calling `mysql_library_end()`.

The purpose of calling `mysql_library_init()` and `mysql_library_end()` is to provide proper initialization and finalization of the MySQL library. For applications that are linked with the client library, they provide improved memory management. If you do not call `mysql_library_end()`, a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.) For applications that are linked with the embedded server, these calls start and stop the server.

`mysql_library_init()` and `mysql_library_end()` are available as of MySQL 4.1.10. For older versions of MySQL, you can call `mysql_server_init()` and `mysql_server_end()` instead.

In a nonmulti-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. This should be done prior to any other client library call.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the host name, user name, and password). Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1`. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL statements to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (for example, `INSERT`, `UPDATE`, `DELETE`), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`, `DESCRIBE`, and `EXPLAIN`. They should be treated the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Client programs should choose the approach that is most appropriate for their requirements. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you do not have random access to rows within the result set (you can only access rows sequentially), and the number of rows in the result set is unknown until you have retrieved them all. Furthermore, you *must* retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to statements (retrieving rows only as necessary) without knowing whether the statement is a `SELECT`. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the statement was a `SELECT` and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether a result was actually to be expected. If `mysql_field_count()` returns zero, the statement returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, and so forth), and was not expected to return rows. If `mysql_field_count()` is nonzero, the statement should have returned rows, but did not. This indicates that the statement was a `SELECT` that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` enable you to obtain information about the fields that make up the result set (the number of fields, their names and types, and so forth). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, MySQL provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the most recently invoked function that can succeed or fail, enabling you to determine when an error occurred and what it was.

17.6.6 C API Function Descriptions

In the descriptions here, a parameter or return value of `NULL` means `NULL` in the sense of the C programming language, not a MySQL `NULL` value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-`NULL` value to indicate success or a `NULL` value to indicate an error, and functions returning an integer return zero to indicate success or nonzero to indicate an error. Note that "nonzero" means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                  /* correct */
    ... error ...

if (result < 0)              /* incorrect */
    ... error ...

if (result == -1)            /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

17.6.6.1 `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

`mysql_affected_rows()` may be called immediately after executing a statement with `mysql_query()` or `mysql_real_query()`. It returns the number of rows changed, deleted, or inserted by the last statement if it was an `UPDATE`, `DELETE`, or `INSERT`. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to `mysqld`, the affected-rows value is the number of rows "found"; that is, matched by the `WHERE` clause.

For `REPLACE` statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For `INSERT ... ON DUPLICATE KEY UPDATE` statements, the affected-rows value is 1 if the row is inserted as a new row and 2 if an existing row is updated.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE` statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`.

Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

Errors

None.

Example

```
char *stmt = "UPDATE products SET cost=cost*1.25
              WHERE group=10";
mysql_query(&mysql,stmt);
printf("%ld products updated",
       (long) mysql_affected_rows(&mysql));
```

17.6.6.2 mysql_autocommit()

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

This function was added in MySQL 4.1.0.

Return Values

Zero if successful. Nonzero if an error occurred.

Errors

None.

17.6.6.3 mysql_change_user()

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password,
const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

This function was introduced in MySQL 3.23.3.

`mysql_change_user()` fails if the connected user cannot be authenticated or does not have permission to use the database. In this case, the user and database are not changed.

The `db` parameter may be set to `NULL` if you do not want to have a default database.

Starting from MySQL 4.0.6, this command resets the state as if one had done a new connect. (See Section 17.6.14, “Controlling Automatic Reconnection Behavior”.) It always performs a `ROLLBACK` of any active transactions, closes and drops all temporary tables, and unlocks all locked tables. Session system variables are reset to the values of the corresponding global system variables. Prepared statements are released and `HANDLER` variables are closed. Locks acquired with `GET_LOCK()` [877] are released. These effects occur even if the user did not change.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`.

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR [1588]

  The MySQL server has gone away.

- CR_SERVER_LOST [1588]

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR [1587]

  An unknown error occurred.

- ER_UNKNOWN_COM_ERROR [1570]

  The MySQL server does not implement this command (probably an old server).

- ER_ACCESS_DENIED_ERROR [1570]

  The user or password was wrong.

- ER_BAD_DB_ERROR [1570]

  The database did not exist.

- ER_DBACCESS_DENIED_ERROR [1570]

  The user did not have access rights to the database.

- ER_WRONG_DB_NAME [1574]

  The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error(&mysql));
}
```

17.6.6.4 mysql_character_set_name()

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set name for the current connection.

Return Values

The default character set name

Errors

None.

17.6.6.5 mysql_close()

```
void mysql_close(MYSQL *mysql)
```

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_connect()`.

Return Values

None.

Errors

None.

17.6.6.6 mysql_commit()

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

This function was added in MySQL 4.1.0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

17.6.6.7 mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const
char *passwd)
```

Description

This function is deprecated. Use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you cannot retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

17.6.6.8 `mysql_create_db()`

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `CREATE DATABASE` statement instead.

Return Values

Zero if the database was created successfully. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error(&mysql));
}
```

17.6.6.9 `mysql_data_seek()`

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a query result set. The `offset` value is a row number and should be in the range from `0` to `mysql_num_rows(result)-1`.

This function requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

None.

Errors

None.

17.6.6.10 `mysql_debug()`

```
void mysql_debug(const char *debug)
```

Description

Does a `DBUG_PUSH` with the given string. `mysql_debug()` uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See Section 18.4.3, "The DBUG Package".

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in `/tmp/client.trace` on the client machine:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

17.6.6.11 `mysql_drop_db()`

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `DROP DATABASE` statement instead.

Return Values

Zero if the database was dropped successfully. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR [1588]

  The MySQL server has gone away.

- CR_SERVER_LOST [1588]

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR [1587]

  An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

17.6.6.12 mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write debugging information to the error log. The connected user must have the SUPER [493] privilege.

Return Values

Zero if the command was successful. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC [1588]

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR [1588]

  The MySQL server has gone away.

- CR_SERVER_LOST [1588]

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR [1587]

  An unknown error occurred.

17.6.6.13 mysql_eof()

```
my_bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. mysql_errno() or mysql_error() may be used instead.

`mysql_eof()` determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a `NULL` return from `mysql_fetch_row()` always means the end of the result set has been reached and it is unnecessary to call `mysql_eof()`. When used with `mysql_store_result()`, `mysql_eof()` always returns true.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()` does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a nonzero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero if no error occurred. Nonzero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

17.6.6.14 `mysql_errno()`

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix B, *Errors, Error Codes, and Common Problems*.

Note that some functions like `mysql_fetch_row()` do not set `mysql_errno()` if they succeed.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_errno()` if they succeed.

MySQL-specific error numbers returned by `mysql_errno()` differ from SQLSTATE values returned by `mysql_sqlstate()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

17.6.6.15 `mysql_error()`

```
const char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function did not fail, the return value of `mysql_error()` may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_error()` if they succeed.

For functions that reset `mysql_error()`, either of these two tests can be used to check for an error:

```
if(*mysql_error(&mysql))
{
  // an error occurred
}

if(mysql_error(&mysql)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages. See Section 9.3, “Setting the Error Message Language”.

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

17.6.6.16 mysql_escape_string()

You should use `mysql_real_escape_string()` instead!

This function is identical to `mysql_real_escape_string()` except that `mysql_real_escape_string()` takes a connection handler as its first argument and escapes the string according to the current character set. `mysql_escape_string()` does not take a connection argument and does not respect the current character set.

17.6.6.17 mysql_fetch_field()

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, MySQL returns the default blob length (8KB) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8KB size is chosen because MySQL does not know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return Values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

17.6.6.18 mysql_fetch_field_direct()

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)
```

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. You may use this function to retrieve the definition for an arbitrary column. The value of `fieldnr` should be in the range from 0 to `mysql_num_fields(result)-1`.

Return Values

The `MYSQL_FIELD` structure for the specified column.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

17.6.6.19 mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
```

```
}
```

17.6.6.20 mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). `NULL` if an error occurred.

Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
         printf("Column %u is %lu bytes in length.\n",
                i, lengths[i]);
    }
}
```

17.6.6.21 mysql_fetch_row()

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Description

Retrieves the next row of a result set. When used after `mysql_store_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve. When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)-1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is `NULL`, the field is `NULL`; otherwise, the field is empty.

Return Values

A `MYSQL_ROW` structure for the next row. `NULL` if there are no more rows to retrieve or if an error occurred.

Errors

Note that error is not reset between calls to `mysql_fetch_row()`

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i < num_fields; i++)
   {
       printf("[%.*s] ", (int) lengths[i],
              row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

17.6.6.22 `mysql_field_count()`

```
unsigned int mysql_field_count(MYSQL *mysql)
```

If you are using a version of MySQL earlier than 3.22.24, you should use `unsigned int mysql_num_fields(MYSQL *mysql)` instead.

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 17.6.13.1, “Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

17.6.6.23 `mysql_field_seek()`

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET
offset)
```

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

17.6.6.24 mysql_field_tell()

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

Errors

None.

17.6.6.25 mysql_free_result()

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, and so forth. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

17.6.6.26 mysql_get_client_info()

```
const char *mysql_get_client_info(void)
```

Description

Returns a string that represents the client library version.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

17.6.6.27 mysql_get_client_version()

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the client library version. The value has the format XYYZZ where X is the major version, YY is the release level, and ZZ is the version number within the release level. For example, a value of 40102 represents a client library version of 4.1.2.

This function was added in MySQL 4.0.16.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

17.6.6.28 mysql_get_host_info()

```
const char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server host name.

Return Values

A character string representing the server host name and the connection type.

Errors

None.

17.6.6.29 mysql_get_proto_info()

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

17.6.6.30 mysql_get_server_info()

```
const char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the server version number.

Return Values

A character string that represents the server version number.

Errors

None.

17.6.6.31 mysql_get_server_version()

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns the version number of the server as an integer.

This function was added in MySQL 4.1.0.

Return Values

A number that represents the MySQL server version in this format:

```
major_version*10000 + minor_version *100 + sub_version
```

For example, 4.1.2 is returned as 40102.

This function is useful in client programs for quickly determining whether some version-specific server capability exists.

Errors

None.

17.6.6.32 mysql_hex_string()

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long
length)
```

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See Section 8.1.1, “String Literals”.

The string in `from` is encoded to hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in `to` and a terminating null byte is appended.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

The return value can be placed into an SQL statement using either `0x*value*` or `X'*value*'` format. However, the return value does not include the `0x` or `X'...'`. The caller must supply whichever of those is desired.

`mysql_hex_string()` was added in MySQL 4.0.23 and 4.1.8.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What is this",12);
end = strmov(end,",0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

17.6.6.33 `mysql_info()`

```
const char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed statement, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of statement, as described here. The numbers are illustrative only; the string contains values appropriate for the statement.

- `INSERT INTO ... SELECT ...`

  String format: `Records: 100 Duplicates: 0 Warnings: 0`

- `INSERT INTO ... VALUES (...),(...),(...)...`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `LOAD DATA INFILE ...`

  String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

- `ALTER TABLE`

String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `UPDATE`

  String format: `Rows matched: 40 Changed: 40 Warnings: 0`

Note that `mysql_info()` returns a non-`NULL` value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed statement. `NULL` if no information is available for the statement.

Errors

None.

17.6.6.34 `mysql_init()`

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a `MYSQL` object suitable for `mysql_real_connect()`. If `mysql` is a `NULL` pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it is freed when `mysql_close()` is called to close the connection.

Return Values

An initialized `MYSQL*` handle. `NULL` if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, `NULL` is returned.

17.6.6.35 `mysql_insert_id()`

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement. Use this function after you have performed an `INSERT` statement into a table that contains an `AUTO_INCREMENT` field, or have used `INSERT` or `UPDATE` to set a column value with `LAST_INSERT_ID(`*`expr`*`)` [874].

More precisely, `mysql_insert_id()` is updated under these conditions:

- `INSERT` statements that store a value into an `AUTO_INCREMENT` column. This is true whether the value is automatically generated by storing the special values `NULL` or `0` into the column, or is an explicit nonspecial value.
- In the case of a multiple-row `INSERT` statement, `mysql_insert_id()` returns the *first* automatically generated `AUTO_INCREMENT` value; if no such value is generated, it returns the *last* explicit value inserted into the `AUTO_INCREMENT` column.

- `INSERT` statements that generate an `AUTO_INCREMENT` value by inserting `LAST_INSERT_ID(expr)` [874] into any column or by updating any column to `LAST_INSERT_ID(expr)` [874].
- If the previous statement returned an error, the value of `mysql_insert_id()` is undefined.

`mysql_insert_id()` returns `0` if the previous statement does not use an `AUTO_INCREMENT` value. If you need to save the value for later, be sure to call `mysql_insert_id()` immediately after the statement that generates the value.

The value of `mysql_insert_id()` is not affected by statements such as `SELECT` that return a result set.

The value of `mysql_insert_id()` is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

The `LAST_INSERT_ID()` [874] SQL function returns the most recently generated `AUTO_INCREMENT` value, and is not reset between statements because the value of that function is maintained in the server. Another difference from `mysql_insert_id()` is that `LAST_INSERT_ID()` [874] is not updated if you set an `AUTO_INCREMENT` column to a specific nonspecial value. See Section 11.13, “Information Functions”.

The reason for the differences between `LAST_INSERT_ID()` [874] and `mysql_insert_id()` is that `LAST_INSERT_ID()` [874] is made easy to use in scripts while `mysql_insert_id()` tries to provide more exact information about what happens to the `AUTO_INCREMENT` column.

Return Values

Described in the preceding discussion.

Errors

None.

17.6.6.36 `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `KILL` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.37 `mysql_library_end()`

```
void mysql_library_end(void)
```

Description

This function finalizes the MySQL library. You should call it when you are done using the library (for example, after disconnecting from the server). The action taken by the call depends on whether your application is linked to the MySQL client library or the MySQL embedded server library. For a client program linked against the `libmysqlclient` library by using the `-lmysqlclient` flag, `mysql_library_end()` performs some memory management to clean up. For an embedded server application linked against the `libmysqld` library by using the `-lmysqld` flag, `mysql_library_end()` shuts down the embedded server and then cleans up.

For usage information, see Section 17.6.5, “C API Function Overview”, and Section 17.6.6.38, “`mysql_library_init()`”.

`mysql_library_end()` was added in MySQL 4.1.10. For older versions of MySQL, call `mysql_server_end()` instead.

17.6.6.38 `mysql_library_init()`

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

This function should be called to initialize the MySQL library before you call any other MySQL function, whether your application is a regular client program or uses the embedded server. If the application uses the embedded server, this call starts the server and initializes any subsystems (`mysys`, `InnoDB`, and so forth) that the server uses.

After your application is done using the MySQL library, call `mysql_library_end()` to clean up. See Section 17.6.6.37, “`mysql_library_end()`”.

The choice of whether the application operates as a regular client or uses the embedded server depends on whether you use the `libmysqlclient` or `libmysqld` library at link time to produce the final executable. For additional information, see Section 17.6.5, “C API Function Overview”.

In a nonmulti-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. This should be done prior to any other client library call.

The `argc` and `argv` arguments are analogous to the arguments to `main()`, and enable passing of options to the embedded server. For convenience, `argc` may be `0` (zero) if there are no command-line arguments

for the server. This is the usual case for applications intended for use only as regular (nonembedded) clients, and the call typically is written as `mysql_library_init(0, NULL, NULL)`.

```
#include <mysql.h>
#include <stdlib.h>

int main(void) {
  if (mysql_library_init(0, NULL, NULL)) {
    fprintf(stderr, "could not initialize MySQL library\n");
    exit(1);
  }

  /* Use any MySQL API functions here */

  mysql_library_end();

  return EXIT_SUCCESS;
}
```

When arguments are to be passed (`argc` is greater than `0`), the first element of `argv` is ignored (it typically contains the program name). `mysql_library_init()` makes a copy of the arguments so it is safe to destroy `argv` or `groups` after the call.

For embedded applications, if you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The `groups` argument should be an array of strings that indicate the groups in option files from which options should be read. See Section 4.2.3.3, “Using Option Files”. The final entry in the array should be `NULL`. For convenience, if the `groups` argument itself is `NULL`, the `[server]` and `[embedded]` groups are used by default.

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
  "this_program",       /* this string is not used */
  "--datadir=.",
  "--key_buffer_size=32M"
};
static char *server_groups[] = {
  "embedded",
  "server",
  "this_program_SERVER",
  (char *)NULL
};

int main(void) {
  if (mysql_library_init(sizeof(server_args) / sizeof(char *),
                         server_args, server_groups)) {
    fprintf(stderr, "could not initialize MySQL library\n");
    exit(1);
  }

  /* Use any MySQL API functions here */

  mysql_library_end();

  return EXIT_SUCCESS;
}
```

`mysql_library_init()` was added in MySQL 4.1.10. For older versions of MySQL, call `mysql_server_init()` instead.

Return Values

Zero if successful. Nonzero if an error occurred.

17.6.6.39 mysql_list_dbs()

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters "`%`" or "`_`", or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW DATABASES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.40 mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Returns an empty result set for which the metadata provides information aobut the columns in the given table that match the simple regular expression specified by the *`wild`* parameter. *`wild`* may contain the wildcard characters "`%`" or "`_`", or may be a `NULL` pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

It is preferable to use `SHOW COLUMNS FROM tbl_name` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

```
int i;
MYSQL_RES *tbl_cols = mysql_list_fields(mysql, "mytbl", "f%");

unsigned int field_cnt = mysql_num_fields(tbl_cols);
printf("Number of columns: %d\n", field_cnt);

for (i=0; i < field_cnt; ++i)
{
  /* col describes i-th column of the table */
  MYSQL_FIELD *col = mysql_fetch_field_direct(tbl_cols, i);
  printf ("Column %d: %s\n", i, col->name);
}
mysql_free_result(tbl_cols);
```

17.6.6.41 `mysql_list_processes()`

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by `mysqladmin processlist` or a `SHOW PROCESSLIST` query.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR [1588]

  The MySQL server has gone away.

- CR_SERVER_LOST [1588]

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR [1587]

  An unknown error occurred.

17.6.6.42 mysql_list_tables()

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the wild parameter. wild may contain the wildcard characters "%" or "_", or may be a NULL pointer to match all tables. Calling mysql_list_tables() is similar to executing the query SHOW TABLES [LIKE *wild*].

You must free the result set with mysql_free_result().

mysql_num_rows() is intended for use with statements that return a result set, such as SELECT. For statements such as INSERT, UPDATE, or DELETE, the number of affected rows can be obtained with mysql_affected_rows().

Return Values

A MYSQL_RES result set for success. NULL if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC [1588]

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR [1588]

  The MySQL server has gone away.

- CR_SERVER_LOST [1588]

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR [1587]

  An unknown error occurred.

17.6.6.43 mysql_more_results()

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string.

`mysql_more_results()` true if more results exist from the currently executed statement, in which case the application must call `mysql_next_result()` to fetch the results.

This function was added in MySQL 4.1.0.

Return Values

`TRUE` (1) if more results exist. `FALSE` (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See Section 17.6.15, "C API Support for Multiple Statement Execution", and Section 17.6.6.44, "`mysql_next_result()`".

Errors

None.

17.6.6.44 `mysql_next_result()`

```
int mysql_next_result(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string.

`mysql_next_result()` reads the next statement result and returns a status to indicate whether more results exist. If `mysql_next_result()` returns an error, there are no more results.

Before each call to `mysql_next_result()`, you must call `mysql_free_result()` for the current statement if it is a statement that returned a result set (rather than just a result status).

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next statement. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If `mysql_next_result()` returns an error, no other statements are executed and there are no more results to fetch.

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_next_result()` to advance to the next result.

For an example that shows how to use `mysql_next_result()`, see Section 17.6.15, "C API Support for Multiple Statement Execution".

This function was added in MySQL 4.1.0.

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results

Return Value	Description
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order. For example, if you did not call `mysql_use_result()` for a previous result set.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.45 `mysql_num_fields()`

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

Or:

```
unsigned int mysql_num_fields(MYSQL *mysql)
```

The second form doesn't work on MySQL 3.22.24 or newer. To pass a `MYSQL*` argument, you must use `unsigned int mysql_field_count(MYSQL *mysql)` instead.

Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` or `mysql_use_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 17.6.13.1, “Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
           fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` returns 0. This happens only if something went wrong.

17.6.6.46 `mysql_num_rows()`

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

Return Values

The number of rows in the result set.

Errors

None.

17.6.6.47 `mysql_options()`

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options.

`mysql_options()` should be called after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, `arg` should point to the value of the integer.

The following list describes the possible options, their effect, and how `arg` is used for each option. Several of the options apply only when the application is linked against the `libmysqld` embedded server library and are unused for applications linked against the `libmysqlclient` client library. For option descriptions that indicate `arg` is unused, its value is irrelevant; it is conventional to pass 0.

- `MYSQL_INIT_COMMAND` (argument type: `char *`)

  SQL statement to execute when connecting to the MySQL server. Automatically re-executed if reconnection occurs.

- `MYSQL_OPT_COMPRESS` (argument: not used)

  Use the compressed client/server protocol.

- `MYSQL_OPT_CONNECT_TIMEOUT` (argument type: `unsigned int *`)

  Connect timeout in seconds.

- `MYSQL_OPT_GUESS_CONNECTION` (argument: not used)

  For an application linked against the `libmysqld` embedded server library, this enables the library to guess whether to use the embedded server or a remote server. “Guess” means that if the host name is set and is not `localhost`, it uses a remote server. This behavior is the default. `MYSQL_OPT_USE_EMBEDDED_CONNECTION` and `MYSQL_OPT_USE_REMOTE_CONNECTION` can be used to override it. This option is ignored for applications linked against the `libmysqlclient` client library. Available starting in 4.1.1.

- `MYSQL_OPT_LOCAL_INFILE` (argument type: optional pointer to `unsigned int`)

  If no pointer is given or if pointer points to an `unsigned int` that has a nonzero value, the `LOAD LOCAL INFILE` statement is enabled.

- `MYSQL_OPT_NAMED_PIPE` (argument: not used)

  Use named pipes to connect to a MySQL server on Windows, if the server permits named-pipe connections.

- `MYSQL_OPT_PROTOCOL` (argument type: `unsigned int *`)

  Type of protocol to use. Should be one of the enum values of `mysql_protocol_type` defined in `mysql.h`. Added in 4.1.0.

- `MYSQL_OPT_READ_TIMEOUT` (argument type: `unsigned int *`)

  The timeout in seconds for attempts to read from the server. Each attempt uses this timeout value and there are retries if necessary, so the total effective timeout value is three times the option value. You can set the value so that a lost connection can be detected earlier than the TCP/IP `Close_Wait_Timeout`

value of 10 minutes. This option works only for TCP/IP connections and, prior to MySQL 4.1.22, only for Windows. Added in 4.1.1.

- `MYSQL_SET_CLIENT_IP` (argument type: `char *`)

  For an application linked against the `libmysqld` embedded server library (when `libmysqld` is compiled with authentication support), this means that the user is considered to have connected from the specified IP address (specified as a string) for authentication purposes. This option is ignored for applications linked against the `libmysqlclient` client library. Added in 4.1.1.

- `MYSQL_OPT_USE_EMBEDDED_CONNECTION` (argument: not used)

  For an application linked against the `libmysqld` embedded server library, this forces the use of the embedded server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library. Added in 4.1.1.

- `MYSQL_OPT_USE_REMOTE_CONNECTION` (argument: not used)

  For an application linked against the `libmysqld` embedded server library, this forces the use of a remote server for the connection. This option is ignored for applications linked against the `libmysqlclient` client library. Added in 4.1.1.

- `MYSQL_OPT_USE_RESULT` (argument: not used)

  This option is available beginning with MySQL 4.1.1, but is unused.

- `MYSQL_OPT_WRITE_TIMEOUT` (argument type: `unsigned int *`)

  The timeout in seconds for attempts to write to the server. Each attempt uses this timeout value and there are `net_retry_count` [422] retries if necessary, so the total effective timeout value is `net_retry_count` [422] times the option value. This option works only for TCP/IP connections and, prior to MySQL 4.1.22, only for Windows. Added in 4.1.1.

- `MYSQL_READ_DEFAULT_FILE` (argument type: `char *`)

  Read options from the named option file instead of from `my.cnf`.

- `MYSQL_READ_DEFAULT_GROUP` (argument type: `char *`)

  Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`.

- `MYSQL_REPORT_DATA_TRUNCATION` (argument type: `my_bool *`)

  Enable or disable reporting of data truncation errors for prepared statements using `MYSQL_BIND.error`. (Default: disabled.)

- `MYSQL_SECURE_AUTH` (argument type: `my_bool*`)

  Whether to connect to a server that does not support the improved password hashing available beginning in MySQL 4.1.1. Added in MySQL 4.1.1.

- `MYSQL_SET_CHARSET_DIR` (argument type: `char *`)

  The path name to the directory that contains character set definition files.

- `MYSQL_SET_CHARSET_NAME` (argument type: `char *`)

  The name of the character set to use as the default character set.

- `MYSQL_SHARED_MEMORY_BASE_NAME` (argument type: `char *`)

  The name of the shared-memory object for communication to the server on Windows, if the server supports shared-memory connections. Should have the same value as the `--shared-memory-base-name` [392] option used for the `mysqld` server you want to connect to. Added in 4.1.0.

The `client` group is always read if you use `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP`.

The specified group in the option file may contain the following options.

Option	Description
`character-sets-dir=path`	The directory where character sets are installed.
`compress`	Use the compressed client/server protocol.
`connect-timeout=seconds`	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
`database=db_name`	Connect to this database if no database was specified in the connect command.
`debug`	Debug options.
`default-character-set=charset_name`	The default character set to use.
`disable-local-infile`	Disable use of `LOAD DATA LOCAL`.
`host=host_name`	Default host name.
`init-command=stmt`	Statement to execute when connecting to MySQL server. Automatically re-executed if reconnection occurs.
`interactive-timeout=seconds`	Same as specifying `CLIENT_INTERACTIVE` to `mysql_real_connect()`. See Section 17.6.6.50, “`mysql_real_connect()`”.
`local-infile[={0|1}]`	If no argument or nonzero argument, enable use of `LOAD DATA LOCAL`; otherwise disable.
`max_allowed_packet=bytes`	Maximum size of packet that client can read from server.
`multi-results`	Permit multiple result sets from multiple-statement executions or stored procedures. Added in 4.1.1.
`multi-queries`, `multi-results`	Permit the client to send multiple statements in a single string (separated by “`;`”). Added in 4.1.9.
`password=password`	Default password.
`pipe`	Use named pipes to connect to a MySQL server on NT.
`port=port_num`	Default port number.
`protocol={TCP|SOCKET|PIPE|MEMORY}`	The protocol to use when connecting to the server. (Added in MySQL 4.1)
`return-found-rows`	Tell `mysql_info()` to return found rows instead of updated rows when using `UPDATE`.
`shared-memory-base-name=name`	Shared-memory name to use to connect to server. Added in MySQL 4.1.
`socket=path`	Default socket file.
`ssl-ca=file_name`	Certificate Authority file.

Option	Description
ssl-capath=*path*	Certificate Authority directory.
ssl-cert=*file_name*	Certificate file.
ssl-cipher=*cipher_list*	Permissible SSL ciphers.
ssl-key=*file_name*	Key file.
timeout=*seconds*	Like connect-timeout.
user	Default user.

timeout has been replaced by connect-timeout, but timeout is still supported in MySQL 4.1 for backward compatibility.

For more information about option files, see Section 4.2.3.3, “Using Option Files”.

Return Values

Zero for success. Nonzero if you specify an unknown option.

Example

The following mysql_options() calls request the use of compression in the client/server protocol, cause options to be read from the [odbc] group of option files, and disable transaction autocommit mode:

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
mysql_options(&mysql,MYSQL_INIT_COMMAND,"SET autocommit=0");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

This code requests that the client use the compressed client/server protocol and read the additional options from the odbc section in the my.cnf file.

17.6.6.48 mysql_ping()

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down and auto-reconnect is enabled an attempt to reconnect is made. If the connection is down and auto-reconnect is disabled, mysql_ping() returns an error.

Auto-reconnect is enabled by default.

mysql_ping() can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

If mysql_ping()) does cause a reconnect, there is no explicit indication of it. To determine whether a reconnect occurs, call mysql_thread_id() to get the original connection identifier before calling mysql_ping(), then call mysql_thread_id() again to see whether the identifier has changed.

If reconnect occurs, some characteristics of the connection will have been reset. For details about these characteristics, see Section 17.6.14, “Controlling Automatic Reconnection Behavior”.

Return Values

Zero if the connection to the server is active. Nonzero if an error occurred. A nonzero return does not indicate whether the MySQL server itself is down; the connection might be broken for other reasons such as network problems.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.49 `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

Description

Executes the SQL statement pointed to by the null-terminated string `stmt_str`. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (“`;`”) or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 17.6.15, “C API Support for Multiple Statement Execution”.

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “`\0`” character, which `mysql_query()` interprets as the end of the statement string.)

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See Section 17.6.6.22, “`mysql_field_count()`”.

Return Values

Zero if the statement was successful. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.50 `mysql_real_connect()`

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char *unix_socket,
unsigned long client_flag)
```

Description

`mysql_real_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_real_connect()` must complete successfully before you can execute any other API functions that require a valid `MYSQL` connection handle structure.

The parameters are specified as follows:

- The first parameter should be the address of an existing `MYSQL` structure. Before calling `mysql_real_connect()` you must call `mysql_init()` to initialize the `MYSQL` structure. You can change a lot of connect options with the `mysql_options()` call. See Section 17.6.6.47, “`mysql_options()`”.
- The value of `host` may be either a host name or an IP address. If `host` is `NULL` or the string `"localhost"`, a connection to the local host is assumed. For Windows, the client connects using a shared-memory connection, if the server has shared-memory connections enabled. Otherwise, TCP/IP is used. For Unix, the client connects using a Unix socket file. For local connections, you can also influence the type of connection to use with the `MYSQL_OPT_PROTOCOL` or `MYSQL_OPT_NAMED_PIPE` options to `mysql_options()`. The type of connection must be supported by the server. For a `host` value of `"."` on Windows, the client connects using a named pipe, if the server has named-pipe connections enabled. If named-pipe connections are not enabled, an error occurs.
- The `user` parameter contains the user's MySQL login ID. If `user` is `NULL` or the empty string `""`, the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current user name must be specified explicitly. See the MyODBC section of Chapter 17, *Connectors and APIs*.
- The `passwd` parameter contains the password for `user`. If `passwd` is `NULL`, only entries in the `user` table for the user that have a blank (empty) password field are checked for a match. This enables the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether they have specified a password.

  Note

  Do not attempt to encrypt the password before calling `mysql_real_connect()`; password encryption is handled automatically by the client API.

- The `user` and `passwd` parameters use whatever character set has been configured for the `MYSQL` object. By default, this is `latin1`, but can be changed by calling `mysql_options(mysql, MYSQL_SET_CHARSET_NAME, "charset_name")` prior to connecting.
- `db` is the database name. If `db` is not `NULL`, the connection sets the default database to this value.
- If `port` is not 0, the value is used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.

- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe that should be used. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags to enable certain features.

Flag Name	Flag Description
`CLIENT_COMPRESS`	Use compression protocol.
`CLIENT_FOUND_ROWS`	Return the number of found (matched) rows, not the number of changed rows.
`CLIENT_IGNORE_SIGPIPE`	Prevents the client library from installing a `SIGPIPE` signal handler. This can be used to avoid conflicts with a handler that the application has already installed.
`CLIENT_IGNORE_SPACE`	Permit spaces after function names. Makes all functions names reserved words.
`CLIENT_INTERACTIVE`	Permit `interactive_timeout` [414] seconds (instead of `wait_timeout` [434] seconds) of inactivity before closing the connection. The client's session `wait_timeout` [434] variable is set to the value of the session `interactive_timeout` [414] variable.
`CLIENT_LOCAL_FILES`	Enable `LOAD DATA LOCAL` handling.
`CLIENT_MULTI_RESULTS`	Tell the server that the client can handle multiple result sets from multiple-statement executions. This is automatically set if `CLIENT_MULTI_STATEMENTS` is set. See the note following this table for more information about this flag. Added in MySQL 4.1.
`CLIENT_MULTI_STATEMENTS`	Tell the server that the client may send multiple statements in a single string (separated by "`;`"). If this flag is not set, multiple-statement execution is disabled. See the note following this table for more information about this flag. Added in MySQL 4.1.
`CLIENT_NO_SCHEMA`	Do not permit the *`db_name.tbl_name.col_name`* syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
`CLIENT_ODBC`	Unused.
`CLIENT_SSL`	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the client library. Instead, use `mysql_ssl_set()` before calling `mysql_real_connect()`.
`CLIENT_REMEMBER_OPTIONS`	Remember options specified by calls to `mysql_options()`. Without this option, if `mysql_real_connect()` fails, you must repeat the `mysql_options()` calls before trying to connect again. With this option, the `mysql_options()` calls need not be repeated. This option was added in MySQL 4.1.2.

If you enable `CLIENT_MULTI_STATEMENTS` or `CLIENT_MULTI_RESULTS`, you should process the result for every call to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 17.6.15, "C API Support for Multiple Statement Execution".

For some parameters, it is possible to have the value taken from an option file rather than from an explicit value in the `mysql_real_connect()` call. To do this, call `mysql_options()` with the `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP` option before calling

`mysql_real_connect()`. Then, in the `mysql_real_connect()` call, specify the “no-value” value for each parameter to be read from an option file:

- For `host`, specify a value of `NULL` or the empty string (`""`).
- For `user`, specify a value of `NULL` or the empty string.
- For `passwd`, specify a value of `NULL`. (For the password, a value of the empty string in the `mysql_real_connect()` call cannot be overridden in an option file, because the empty string indicates explicitly that the MySQL account must have an empty password.)
- For `db`, specify a value of `NULL` or the empty string.
- For `port`, specify a value of 0.
- For `unix_socket`, specify a value of `NULL`.

If no value is found in an option file for a parameter, its default value is used as indicated in the descriptions given earlier in this section.

Return Values

A `MYSQL*` connection handle if the connection was successful, `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- `CR_CONN_HOST_ERROR` [1588]

  Failed to connect to the MySQL server.

- `CR_CONNECTION_ERROR` [1588]

  Failed to connect to the local MySQL server.

- `CR_IPSOCK_ERROR` [1588]

  Failed to create an IP socket.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SOCKET_CREATE_ERROR` [1587]

  Failed to create a Unix socket.

- `CR_UNKNOWN_HOST` [1588]

  Failed to find the IP address for the host name.

- `CR_VERSION_ERROR` [1588]

  A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version. This can happen if you use a very old client library to connect to a new server that wasn't started with the `--old-protocol` [391] option.

- `CR_NAMEDPIPEOPEN_ERROR` [1589]

Failed to create a named pipe on Windows.

- `CR_NAMEDPIPEWAIT_ERROR` [1588]

  Failed to wait for a named pipe on Windows.

- `CR_NAMEDPIPESETSTATE_ERROR` [1589]

  Failed to get a pipe handler on Windows.

- `CR_SERVER_LOST` [1588]

  If `connect_timeout` [409] > 0 and it took longer than `connect_timeout` [409] seconds to connect to the server or if the server died while executing the `init-command`.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

By using `mysql_options()` the MySQL library reads the `[client]` and `[your_prog_name]` sections in the `my.cnf` file which ensures that your program works, even if someone has set up MySQL in some nonstandard way.

Note that upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1`. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up.

17.6.6.51 `mysql_real_escape_string()`

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char
*from, unsigned long length)
```

Note that `mysql` must be a valid, open connection. This is needed because the escaping depends on the character set in use by the server.

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See Section 8.1.1, “String Literals”.

The string in `from` is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in `to` and a terminating null byte is appended. Characters encoded are `NUL` (ASCII 0), “`\n`”, “`\r`”, “`\`”, “`'`”, “`"`”, and Control-Z (see Section 8.1, “Literal Values”). (Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. This function quotes the other characters to make them easier to read in log files.)

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character may need to be encoded as using two bytes, and you need room for the terminating null byte.) When `mysql_real_escape_string()` returns, the

contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

If you need to change the character set of the connection, you should use the `mysql_set_character_set()` function rather than executing a `SET NAMES` (or `SET CHARACTER SET`) statement. `mysql_set_character_set()` works like `SET NAMES` but also affects the character set used by `mysql_real_escape_string()`, which `SET NAMES` does not.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"What is this",12);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

17.6.6.52 `mysql_real_query()`

```
int mysql_real_query(MYSQL *mysql, const char *stmt_str, unsigned long length)
```

Description

Executes the SQL statement pointed to by `stmt_str`, which should be a string `length` bytes long. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (“`;`”) or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 17.6.15, “C API Support for Multiple Statement Execution”.

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “`\0`” character, which `mysql_query()` interprets as the end of the statement string.) In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the statement string.

If you want to know whether the statement should return a result set, you can use `mysql_field_count()` to check for this. See Section 17.6.6.22, “`mysql_field_count()`”.

Return Values

Zero if the statement was successful. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.53 `mysql_refresh()`

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

Description

This function flushes tables or caches, or resets replication server information. The connected user must have the `RELOAD` [492] privilege.

The `options` argument is a bit mask composed from any combination of the following values. Multiple values can be OR'ed together to perform multiple operations with a single call.

- `REFRESH_GRANT`

  Refresh the grant tables, like `FLUSH PRIVILEGES`.

- `REFRESH_LOG`

  Flush the logs, like `FLUSH LOGS`.

- `REFRESH_TABLES`

  Flush the table cache, like `FLUSH TABLES`.

- `REFRESH_HOSTS`

  Flush the host cache, like `FLUSH HOSTS`.

- `REFRESH_STATUS`

  Reset status variables, like `FLUSH STATUS`.

- `REFRESH_THREADS`

  Flush the thread cache.

- `REFRESH_SLAVE`

On a slave replication server, reset the master server information and restart the slave, like `RESET SLAVE`.

- `REFRESH_MASTER`

  On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like `RESET MASTER`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.54 `mysql_reload()`

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the `RELOAD` [492] privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `FLUSH PRIVILEGES` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.55 `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

This function was added in MySQL 4.1.0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

17.6.6.56 `mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The `offset` value is a row offset that should be a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

17.6.6.57 `mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

You should use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

17.6.6.58 `mysql_select_db()`

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.59 `mysql_set_character_set()`

```
int mysql_set_character_set(MYSQL *mysql, const char *csname)
```

Description

This function is used to set the default character set for the current connection. The string `csname` specifies a valid character set name. The connection collation becomes the default collation of the character set. This function works like the `SET NAMES` statement, but also sets the value of `mysql->charset`, and thus affects the character set used by `mysql_real_escape_string()`

This function was added in MySQL 4.1.13.

Return Values

Zero for success. Nonzero if an error occurred.

Example

```
MYSQL mysql;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n",
           mysql_character_set_name(&mysql));
}
```

17.6.6.60 mysql_set_local_infile_default()

```
void mysql_set_local_infile_default(MYSQL *mysql);
```

Description

Sets the `LOAD LOCAL DATA INFILE` handler callback functions to the defaults used internally by the C client library. The library calls this function automatically if `mysql_set_local_infile_handler()` has not been called or does not supply valid functions for each of its callbacks.

The `mysql_set_local_infile_default()` function was added in MySQL 4.1.2.

Return Values

None.

Errors

None.

17.6.6.61 mysql_set_local_infile_handler()

```
void mysql_set_local_infile_handler(MYSQL *mysql, int (*local_infile_init)(void
**, const char *, void *), int (*local_infile_read)(void *, char *, unsigned
int), void (*local_infile_end)(void *), int (*local_infile_error)(void *,
char*, unsigned int), void *userdata);
```

Description

This function installs callbacks to be used during the execution of `LOAD DATA LOCAL INFILE` statements. It enables application programs to exert control over local (client-side) data file reading. The arguments are the connection handler, a set of pointers to callback functions, and a pointer to a data area that the callbacks can use to share information.

To use `mysql_set_local_infile_handler()`, you must write the following callback functions:

```
int
```

```
local_infile_init(void **ptr, const char *filename, void *userdata);
```

The initialization function. This is called once to do any setup necessary, open the data file, allocate data structures, and so forth. The first `void**` argument is a pointer to a pointer. You can set the pointer (that is, `*ptr`) to a value that will be passed to each of the other callbacks (as a `void*`). The callbacks can use this pointed-to value to maintain state information. The `userdata` argument is the same value that is passed to `mysql_set_local_infile_handler()`.

The initialization function should return zero for success, nonzero for an error.

```
int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);
```

The data-reading function. This is called repeatedly to read the data file. `buf` points to the buffer where the read data should be stored, and `buf_len` is the maximum number of bytes that the callback can read and store in the buffer. (It can read fewer bytes, but should not read more.)

The return value is the number of bytes read, or zero when no more data could be read (this indicates EOF). Return a value less than zero if an error occurs.

```
void
local_infile_end(void *ptr)
```

The termination function. This is called once after `local_infile_read()` has returned zero (EOF) or an error. This function should deallocate any memory allocated by `local_infile_init()` and perform any other cleanup necessary. It is invoked even if the initalization function returns an error.

```
int
local_infile_error(void *ptr,
                   char *error_msg,
                   unsigned int error_msg_len);
```

The error-handling function. This is called to get a textual error message to return to the user in case any of your other functions returns an error. `error_msg` points to the buffer into which the message should be written, and `error_msg_len` is the length of the buffer. The message should be written as a null-terminated string, so the message can be at most `error_msg_len`–1 bytes long.

The return value is the error number.

Typically, the other callbacks store the error message in the data structure pointed to by `ptr`, so that `local_infile_error()` can copy the message from there into `error_msg`.

After calling `mysql_set_local_infile_handler()` in your C code and passing pointers to your callback functions, you can then issue a `LOAD DATA LOCAL INFILE` statement (for example, by using `mysql_query()`). The client library automatically invokes your callbacks. The file name specified in `LOAD DATA LOCAL INFILE` will be passed as the second parameter to the `local_infile_init()` callback.

The `mysql_set_local_infile_handler()` function was added in MySQL 4.1.2.

Return Values

None.

Errors

None.

17.6.6.62 mysql_set_server_option()

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. `option` can have one of the following values.

Option	Description
`MYSQL_OPTION_MULTI_STATEMENTS_ON`	Enable multiple-statement support
`MYSQL_OPTION_MULTI_STATEMENTS_OFF`	Disable multiple-statement support

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 17.6.15, “C API Support for Multiple Statement Execution”.

Enabling multiple-statement support with `MYSQL_OPTION_MULTI_STATEMENTS_ON` does not have quite the same effect as enabling it by passing the `CLIENT_MULTI_STATEMENTS` flag to `mysql_real_connect()`: `CLIENT_MULTI_STATEMENTS` also enables `CLIENT_MULTI_RESULTS`.

This function was added in MySQL 4.1.1.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `ER_UNKNOWN_COM_ERROR` [1570]

  The server did not support `mysql_set_server_option()` (which is the case that the server is older than 4.1.1) or the server did not support the option one tried to set.

17.6.6.63 mysql_shutdown()

```
int mysql_shutdown(MYSQL *mysql, enum mysql_enum_shutdown_level shutdown_level)
```

Description

Asks the database server to shut down. The connected user must have the `SHUTDOWN` [492] privilege. The `shutdown_level` argument was added in MySQL 4.1.3. MySQL 4.1 supports only one type of shutdown; `shutdown_level` must be equal to `SHUTDOWN_DEFAULT`. Additional shutdown levels are planned to make it possible to choose the desired level. Dynamically linked executables which have been compiled

with older versions of the `libmysqlclient` headers and call `mysql_shutdown()` need to be used with the old `libmysqlclient` dynamic library.

The shutdown process is described in Section 5.1.9, “The Shutdown Process”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.64 `mysql_sqlstate()`

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the most recently executed SQL statement. The error code consists of five characters. `'00000'` means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see Appendix B, *Errors, Error Codes, and Common Problems*.

SQLSTATE values returned by `mysql_sqlstate()` differ from MySQL-specific error numbers returned by `mysql_errno()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Not all MySQL error numbers are mapped to SQLSTATE error codes. The value `'HY000'` (general error) is used for unmapped error numbers.

If you call `mysql_sqlstate()` after `mysql_real_connect()` fails, `mysql_sqlstate()` might not return a useful value. For example, this happens if a host is blocked by the server and the connection is closed without any SQLSTATE value being sent to the client.

This function was added in MySQL 4.1.1.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See Section 17.6.6.14, “mysql_errno()”, Section 17.6.6.15, “mysql_error()”, and Section 17.6.10.26, “mysql_stmt_sqlstate()”.

17.6.6.65 mysql_ssl_set()

```
my_bool mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const
char *ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing secure connections using SSL. It must be called before `mysql_real_connect()`.

`mysql_ssl_set()` does nothing unless SSL support is enabled in the client library.

`mysql` is the connection handler returned from `mysql_init()`. The other parameters are specified as follows:

- `key` is the path name to the key file.
- `cert` is the path name to the certificate file.
- `ca` is the path name to the certificate authority file.
- `capath` is the path name to a directory that contains trusted SSL CA certificates in pem format.
- `cipher` is a list of permissible ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`.

Return Values

This function always returns `0`. If SSL setup is incorrect, `mysql_real_connect()` returns an error when you attempt to connect.

17.6.6.66 mysql_stat()

```
const char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR [1588]

  The MySQL server has gone away.

- CR_SERVER_LOST [1588]

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR [1587]

  An unknown error occurred.

17.6.6.67 mysql_store_result()

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

After invoking mysql_query() or mysql_real_query(), you must call mysql_store_result() or mysql_use_result() for every statement that successfully produces a result set (SELECT, SHOW, DESCRIBE, EXPLAIN, CHECK TABLE, and so forth). You must also call mysql_free_result() after you are done with the result set.

You need not call mysql_store_result() or mysql_use_result() for other statements, but it does not do any harm or cause any notable performance degradation if you call mysql_store_result() in all cases. You can detect whether the statement has a result set by checking whether mysql_store_result() returns a nonzero value (more about this later on).

If you enable multiple-statement support, you should retrieve results from calls to mysql_query() or mysql_real_query() by using a loop that calls mysql_next_result() to determine whether there are more results. For an example, see Section 17.6.15, “C API Support for Multiple Statement Execution”.

If you want to know whether a statement should return a result set, you can use mysql_field_count() to check for this. See Section 17.6.6.22, “mysql_field_count()”.

mysql_store_result() reads the entire result of a query to the client, allocates a MYSQL_RES structure, and places the result into this structure.

mysql_store_result() returns a null pointer if the statement did not return a result set (for example, if it was an INSERT statement).

mysql_store_result() also returns a null pointer if reading of the result set failed. You can check whether an error occurred by checking whether mysql_error() returns a nonempty string, mysql_errno() returns nonzero, or mysql_field_count() returns zero.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

After you have called mysql_store_result() and gotten back a result that is not a null pointer, you can call mysql_num_rows() to find out how many rows are in the result set.

You can call mysql_fetch_row() to fetch rows from the result set, or mysql_row_seek() and mysql_row_tell() to obtain or set the current row position within the result set.

See Section 17.6.13.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

Return Values

A `MYSQL_RES` result structure with the results. `NULL` (0) if an error occurred.

Errors

`mysql_store_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.68 `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.

Return Values

The thread ID of the current connection.

Errors

None.

17.6.6.69 `mysql_use_result()`

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client allocates memory only for the current row and a communication buffer that may grow up to `max_allowed_packet` [418] bytes.

On the other hand, you should not use `mysql_use_result()` if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise, the unfetched rows are returned as part of the result set for your next query. The C API gives the error `Commands out of sync; you can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` accurately returns the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

When using the `libmysqld` embedded server, the memory benefits are essentially lost because memory usage incrementally increases with each row retrieved until `mysql_free_result()` is called.

Return Values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`mysql_use_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.6.70 `mysql_warning_count()`

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of warnings generated during execution of the previous SQL statement.

This function was added in MySQL 4.1.0.

Return Values

The warning count.

Errors

None.

17.6.7 C API Prepared Statements

As of MySQL 4.1, the client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handle returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Prepared statements might not provide a performance increase in some situations. For best results, test your application both with prepared and nonprepared statements and choose whichever yields best performance.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

For a list of SQL statements that can be used as prepared statements, see Section 12.6, “SQL Syntax for Prepared Statements”.

17.6.8 C API Prepared Statement Data Structures

Note

Some incompatible changes were made in MySQL 4.1.2. See Section 17.6.10, “C API Prepared Statement Function Descriptions”, for details.

Prepared statements use several data structures:

- To obtain a statement handle, pass a `MYSQL` connection handler to `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT` data structure. This structure is used for further operations with the statement. To specify the statement to prepare, pass the `MYSQL_STMT` pointer and the statement string to `mysql_stmt_prepare()`.
- To provide input parameters for a prepared statement, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_param()`. To receive output column values, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_result()`.

- The `MYSQL_TIME` structure is used to transfer temporal data in both directions.

The following discussion describes the prepared statement data types in detail. For examples that show how to use them, see Section 17.6.10.10, “`mysql_stmt_execute()`”, and Section 17.6.10.11, “`mysql_stmt_fetch()`”.

- `MYSQL_STMT`

  This structure is a handle for a prepared statement. A handle is created by calling `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT`. The handle is used for all subsequent operations with the statement until you close it with `mysql_stmt_close()`, at which point the handle becomes invalid.

  The `MYSQL_STMT` structure has no members intended for application use. Applications should not try to copy a `MYSQL_STMT` structure. There is no guarantee that such a copy will be usable.

  Multiple statement handles can be associated with a single connection. The limit on the number of handles depends on the available system resources.

- `MYSQL_BIND`

  This structure is used both for statement input (data values sent to the server) and output (result values returned from the server):

  - For input, use `MYSQL_BIND` structures with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`.
  - For output, use `MYSQL_BIND` structures with `mysql_stmt_bind_result()` to bind buffers to result set columns, for use in fetching rows with `mysql_stmt_fetch()`.

  To use a `MYSQL_BIND` structure, zero its contents to initialize it, then set its members appropriately. For example, to declare and initialize an array of three `MYSQL_BIND` structures, use this code:

```
MYSQL_BIND bind[3];
memset(bind, 0, sizeof(bind));
```

  The `MYSQL_BIND` structure contains the following members for use by application programs. For several of the members, the manner of use depends on whether the structure is used for input or output.

  - `enum enum_field_types buffer_type`

    The type of the buffer. This member indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored. For permissible `buffer_type` values, see Section 17.6.8.1, “C API Prepared Statement Type Codes”.

  - `void *buffer`

    A pointer to the buffer to be used for data transfer. This is the address of a C language variable.

    For input, `buffer` is a pointer to the variable in which you store the data value for a statement parameter. When you call `mysql_stmt_execute()`, MySQL use the value stored in the variable in place of the corresponding parameter marker in the statement (specified with `?` in the statement string).

For output, `buffer` is a pointer to the variable in which to return a result set column value. When you call `mysql_stmt_fetch()`, MySQL stores a column value from the current row of the result set in this variable. You can access the value when the call returns.

To minimize the need for MySQL to perform type conversions between C language values on the client side and SQL values on the server side, use C variables that have types similar to those of the corresponding SQL values:

- For numeric data types, `buffer` should point to a variable of the proper numeric C type. For integer variables (which can be `char` for single-byte values or an integer type for larger values), you should also indicate whether the variable has the `unsigned` attribute by setting the `is_unsigned` member, described later.
- For character (nonbinary) and binary string data types, `buffer` should point to a character buffer.
- For date and time data types, `buffer` should point to a `MYSQL_TIME` structure.

For guidelines about mapping between C types and SQL types and notes about type conversions, see Section 17.6.8.1, “C API Prepared Statement Type Codes”, and Section 17.6.8.2, “C API Prepared Statement Type Conversions”.

- `unsigned long buffer_length`

  The actual size of `*buffer` in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the `buffer_length` value specifies the length of `*buffer` when used with `mysql_stmt_bind_param()` to specify input values, or the maximum number of output data bytes that can be fetched into the buffer when used with `mysql_stmt_bind_result()`.

- `unsigned long *length`

  A pointer to an `unsigned long` variable that indicates the actual number of bytes of data stored in `*buffer`. `length` is used for character or binary C data.

  For input parameter data binding, set `*length` to indicate the actual length of the parameter value stored in `*buffer`. This is used by `mysql_stmt_execute()`.

  For output value binding, MySQL sets `*length` when you call `mysql_stmt_fetch()`. The `mysql_stmt_fetch()` return value determines how to interpret the length:

  - If the return value is 0, `*length` indicates the actual length of the parameter value.
  - If the return value is `MYSQL_DATA_TRUNCATED`, `*length` indicates the nontruncated length of the parameter value. In this case, the minimum of `*length` and `buffer_length` indicates the actual length of the value.

  `length` is ignored for numeric and temporal data types because the `buffer_type` value determines the length of the data value.

  If you must determine the length of a returned value before fetching it, see Section 17.6.10.11, “`mysql_stmt_fetch()`”, for some strategies.

- `my_bool *is_null`

  This member points to a `my_bool` variable that is true if a value is `NULL`, false if it is not `NULL`. For input, set `*is_null` to true to indicate that you are passing a `NULL` value as a statement parameter.

`is_null` is a *pointer* to a boolean scalar, not a boolean scalar, to provide flexibility in how you specify `NULL` values:

- If your data values are always `NULL`, use `MYSQL_TYPE_NULL` as the `buffer_type` value when you bind the column. The other `MYSQL_BIND` members, including `is_null`, do not matter.
- If your data values are always `NOT NULL`, set `is_null = (my_bool*) 0`, and set the other members appropriately for the variable you are binding.
- In all other cases, set the other members appriopriately and set `is_null` to the address of a `my_bool` variable. Set that variable's value to true or false appropriately between executions to indicate whether the corresponding data value is `NULL` or `NOT NULL`, respectively.

For output, when you fetch a row, MySQL sets the value pointed to by `is_null` to true or false according to whether the result set column value returned from the statement is or is not `NULL`.

- `my_bool is_unsigned`

  This member applies for C variables with data types that can be `unsigned` (`char`, `short int`, `int`, `long long int`). Set `is_unsigned` to true if the variable pointed to by `buffer` is `unsigned` and false otherwise. For example, if you bind a `signed char` variable to `buffer`, specify a type code of `MYSQL_TYPE_TINY` and set `is_unsigned` to false. If you bind an `unsigned char` instead, the type code is the same but `is_unsigned` should be true. (For `char`, it is not defined whether it is signed or unsigned, so it is best to be explicit about signedness by using `signed char` or `unsigned char`.)

  `is_unsigned` applies only to the C language variable on the client side. It indicates nothing about the signedness of the corresponding SQL value on the server side. For example, if you use an `int` variable to supply a value for a `BIGINT UNSIGNED` column, `is_unsigned` should be false because `int` is a signed type. If you use an `unsigned int` variable to supply a value for a `BIGINT` column, `is_unsigned` should be true because `unsigned int` is an unsigned type. MySQL performs the proper conversion between signed and unsigned values in both directions, although a warning occurs if truncation results.

- `MYSQL_TIME`

This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. Set the `buffer` member to point to a `MYSQL_TIME` structure, and set the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types (`MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATE`, `MYSQL_TYPE_DATETIME`, `MYSQL_TYPE_TIMESTAMP`).

The `MYSQL_TIME` structure contains the members listed in the following table.

Member	Description
`unsigned int year`	The year
`unsigned int month`	The month of the year
`unsigned int day`	The day of the month
`unsigned int hour`	The hour of the day
`unsigned int minute`	The minute of the hour
`unsigned int second`	The second of the minute
`my_bool neg`	A boolean flag indicating whether the time is negative

Member	Description
`unsigned long second_part`	The fractional part of the second in microseconds; currently unused

Only those parts of a `MYSQL_TIME` structure that apply to a given type of temporal value are used. The `year`, `month`, and `day` elements are used for `DATE`, `DATETIME`, and `TIMESTAMP` values. The `hour`, `minute`, and `second` elements are used for `TIME`, `DATETIME`, and `TIMESTAMP` values. See Section 17.6.17, “C API Prepared Statement Handling of Date and Time Values”.

17.6.8.1 C API Prepared Statement Type Codes

The `buffer_type` member of `MYSQL_BIND` structures indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored.

The following table shows the permissible values for the `buffer_type` member of `MYSQL_BIND` structures for input values sent to the server. The table shows the C variable types that you can use, the corresponding type codes, and the SQL data types for which the supplied value can be used without conversion. Choose the `buffer_type` value according to the data type of the C language variable that you are binding. For the integer types, you should also set the `is_unsigned` member to indicate whether the variable is signed or unsigned.

Input Variable C Type	`buffer_type` Value	SQL Type of Destination Value
`signed char`	`MYSQL_TYPE_TINY`	`TINYINT`
`short int`	`MYSQL_TYPE_SHORT`	`SMALLINT`
`int`	`MYSQL_TYPE_LONG`	`INT`
`long long int`	`MYSQL_TYPE_LONGLONG`	`BIGINT`
`float`	`MYSQL_TYPE_FLOAT`	`FLOAT`
`double`	`MYSQL_TYPE_DOUBLE`	`DOUBLE`
`MYSQL_TIME`	`MYSQL_TYPE_TIME`	`TIME`
`MYSQL_TIME`	`MYSQL_TYPE_DATE`	`DATE`
`MYSQL_TIME`	`MYSQL_TYPE_DATETIME`	`DATETIME`
`MYSQL_TIME`	`MYSQL_TYPE_TIMESTAMP`	`TIMESTAMP`
`char[]`	`MYSQL_TYPE_STRING`	`TEXT`, `CHAR`, `VARCHAR`
`char[]`	`MYSQL_TYPE_BLOB`	`BLOB`, `BINARY`, `VARBINARY`
	`MYSQL_TYPE_NULL`	`NULL`

Use `MYSQL_TYPE_NULL` as indicated in the description for the `is_null` member in Section 17.6.8, “C API Prepared Statement Data Structures”.

For input string data, use `MYSQL_TYPE_STRING` or `MYSQL_TYPE_BLOB` depending on whether the value is a character (nonbinary) or binary string:

- `MYSQL_TYPE_STRING` indicates character input string data. The value is assumed to be in the character set indicated by the `character_set_client` [408] system variable. If the server stores the value into a column with a different character set, it converts the value to that character set.
- `MYSQL_TYPE_BLOB` indicates binary input string data. The value is treated as having the `binary` character set. That is, it is treated as a byte string and no conversion occurs.

The following table shows the permissible values for the `buffer_type` member of `MYSQL_BIND` structures for output values received from the server. The table shows the SQL types of received values, the corresponding type codes that such values have in result set metadata, and the recommended C language data types to bind to the `MYSQL_BIND` structure to receive the SQL values without conversion. Choose the `buffer_type` value according to the data type of the C language variable that you are binding. For the integer types, you should also set the `is_unsigned` member to indicate whether the variable is signed or unsigned.

SQL Type of Received Value	`buffer_type` Value	Output Variable C Type
`TINYINT`	`MYSQL_TYPE_TINY`	`signed char`
`SMALLINT`	`MYSQL_TYPE_SHORT`	`short int`
`MEDIUMINT`	`MYSQL_TYPE_INT24`	`int`
`INT`	`MYSQL_TYPE_LONG`	`int`
`BIGINT`	`MYSQL_TYPE_LONGLONG`	`long long int`
`FLOAT`	`MYSQL_TYPE_FLOAT`	`float`
`DOUBLE`	`MYSQL_TYPE_DOUBLE`	`double`
`YEAR`	`MYSQL_TYPE_SHORT`	`short int`
`TIME`	`MYSQL_TYPE_TIME`	`MYSQL_TIME`
`DATE`	`MYSQL_TYPE_DATE`	`MYSQL_TIME`
`DATETIME`	`MYSQL_TYPE_DATETIME`	`MYSQL_TIME`
`TIMESTAMP`	`MYSQL_TYPE_TIMESTAMP`	`MYSQL_TIME`
`CHAR`, `BINARY`	`MYSQL_TYPE_STRING`	`char[]`
`VARCHAR`, `VARBINARY`	`MYSQL_TYPE_VAR_STRING`	`char[]`
`TINYBLOB`, `TINYTEXT`	`MYSQL_TYPE_TINY_BLOB`	`char[]`
`BLOB`, `TEXT`	`MYSQL_TYPE_BLOB`	`char[]`
`MEDIUMBLOB`, `MEDIUMTEXT`	`MYSQL_TYPE_MEDIUM_BLOB`	`char[]`
`LONGBLOB`, `LONGTEXT`	`MYSQL_TYPE_LONG_BLOB`	`char[]`

17.6.8.2 C API Prepared Statement Type Conversions

Prepared statements transmit data between the client and server using C language variables on the client side that correspond to SQL values on the server side. If there is a mismatch between the C variable type on the client side and the corresponding SQL value type on the server side, MySQL performs implicit type conversions in both directions.

MySQL knows the type code for the SQL value on the server side. The `buffer_type` value in the `MYSQL_BIND` structure indicates the type code of the C variable that holds the value on the client side. The two codes together tell MySQL what conversion must be performed, if any. Here are some examples:

- If you use `MYSQL_TYPE_LONG` with an `int` variable to pass an integer value to the server that is to be stored into a `FLOAT` column, MySQL converts the value to floating-point format before storing it.
- If you fetch an SQL `MEDIUMINT` column value, but specify a `buffer_type` value of `MYSQL_TYPE_LONGLONG` and use a C variable of type `long long int` as the destination buffer, MySQL converts the `MEDIUMINT` value (which requires less than 8 bytes) for storage into the `long long int` (an 8-byte variable).

- If you fetch a numeric column with a value of 255 into a `char[4]` character array and specify a `buffer_type` value of `MYSQL_TYPE_STRING`, the resulting value in the array is a 4-byte string `'255\0'`.
- MySQL returns `DECIMAL` values as the string representation of the original server-side value, which is why the corresponding C type is `char[]`. For example, `12.345` is returned to the client as `'12.345'`. If you specify `MYSQL_TYPE_NEWDECIMAL` and bind a string buffer to the `MYSQL_BIND` structure, `mysql_stmt_fetch()` stores the value in the buffer as a string without conversion. If instead you specify a numeric variable and type code, `mysql_stmt_fetch()` converts the string-format `DECIMAL` value to numeric form.
- For the `MYSQL_TYPE_BIT` type code, `BIT` values are returned into a string buffer, which is why the corresponding C type is `char[]`. The value represents a bit string that requires interpretation on the client side. To return the value as a type that is easier to deal with, you can cause the value to be cast to integer using either of the following types of expressions:

  ```
  SELECT bit_col + 0 FROM t
  SELECT CAST(bit_col AS UNSIGNED) FROM t
  ```

  To retrieve the value, bind an integer variable large enough to hold the value and specify the appropriate corresponding integer type code.

Before binding variables to the `MYSQL_BIND` structures that are to be used for fetching column values, you can check the type codes for each column of the result set. This might be desirable if you want to determine which variable types would be best to use to avoid type conversions. To get the type codes, call `mysql_stmt_result_metadata()` after executing the prepared statement with `mysql_stmt_execute()`. The metadata provides access to the type codes for the result set as described in Section 17.6.10.22, “`mysql_stmt_result_metadata()`”, and Section 17.6.4, “C API Data Structures”.

To determine whether output string values in a result set returned from the server contain binary or nonbinary data, check whether the `charsetnr` value of the result set metadata is 63 (see Section 17.6.4, “C API Data Structures”). If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

If you cause the `max_length` member of the `MYSQL_FIELD` column metadata structures to be set (by calling `mysql_stmt_attr_set()`), be aware that the `max_length` values for the result set indicate the lengths of the longest string representation of the result values, not the lengths of the binary representation. That is, `max_length` does not necessarily correspond to the size of the buffers needed to fetch the values with the binary protocol used for prepared statements. Choose the size of the buffers according to the types of the variables into which you fetch the values. For example, a `TINYINT` column containing the value -128 might have a `max_length` value of 4. But the binary representation of any `TINYINT` value requires only 1 byte for storage, so you can supply a `signed char` variable in which to store the value and set `is_unsigned` to indicate that values are signed.

17.6.9 C API Prepared Statement Function Overview

Note

Some incompatible changes were made in MySQL 4.1.2. See Section 17.6.10, “C API Prepared Statement Function Descriptions”, for details.

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See Section 17.6.10, “C API Prepared Statement Function Descriptions”.

Function	Description
mysql_stmt_affected_rows()	Returns the number of rows changed, deleted, or inserted by prepared UPDATE, DELETE, or INSERT statement
mysql_stmt_attr_get()	Gets value of an attribute for a prepared statement
mysql_stmt_attr_set()	Sets an attribute for a prepared statement
mysql_stmt_bind_param()	Associates application data buffers with the parameter markers in a prepared SQL statement
mysql_stmt_bind_result()	Associates application data buffers with columns in a result set
mysql_stmt_close()	Frees memory used by a prepared statement
mysql_stmt_data_seek()	Seeks to an arbitrary row number in a statement result set
mysql_stmt_errno()	Returns the error number for the last statement execution
mysql_stmt_error()	Returns the error message for the last statement execution
mysql_stmt_execute()	Executes a prepared statement
mysql_stmt_fetch()	Fetches the next row of data from a result set and returns data for all bound columns
mysql_stmt_fetch_column()	Fetch data for one column of the current row of a result set
mysql_stmt_field_count()	Returns the number of result columns for the most recent statement
mysql_stmt_free_result()	Free the resources allocated to a statement handle
mysql_stmt_init()	Allocates memory for a MYSQL_STMT structure and initializes it
mysql_stmt_insert_id()	Returns the ID generated for an AUTO_INCREMENT column by a prepared statement
mysql_stmt_num_rows()	Returns the row count from a buffered statement result set
mysql_stmt_param_count()	Returns the number of parameters in a prepared statement
mysql_stmt_param_metadata()	(Return parameter metadata in the form of a result set) Currently, this function does nothing
mysql_stmt_prepare()	Prepares an SQL statement string for execution
mysql_stmt_reset()	Resets the statement buffers in the server
mysql_stmt_result_metadata()	Returns prepared statement metadata in the form of a result set
mysql_stmt_row_seek()	Seeks to a row offset in a statement result set, using value returned from mysql_stmt_row_tell()
mysql_stmt_row_tell()	Returns the statement row cursor position
mysql_stmt_send_long_data()	Sends long data in chunks to server
mysql_stmt_sqlstate()	Returns the SQLSTATE error code for the last statement execution
mysql_stmt_store_result()	Retrieves a complete result set to the client

Call mysql_stmt_init() to create a statement handle, then mysql_stmt_prepare() to prepare the statement string, mysql_stmt_bind_param() to supply the parameter data, and mysql_stmt_execute() to execute the statement. You can repeat the mysql_stmt_execute() by changing parameter values in the respective buffers supplied through mysql_stmt_bind_param().

You can send text or binary data in chunks to server using mysql_stmt_send_long_data(). See Section 17.6.10.25, “mysql_stmt_send_long_data()”.

If the statement is a `SELECT` or any other statement that produces a result set, `mysql_stmt_prepare()` also returns the result set metadata information in the form of a `MYSQL_RES` result set through `mysql_stmt_result_metadata()`.

You can supply the result buffers using `mysql_stmt_bind_result()`, so that the `mysql_stmt_fetch()` automatically returns data to these buffers. This is row-by-row fetching.

When statement execution has been completed, close the statement handle using `mysql_stmt_close()` so that all resources associated with it can be freed.

If you obtained a `SELECT` statement's result set metadata by calling `mysql_stmt_result_metadata()`, you should also free the metadata using `mysql_free_result()`.

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handle with `mysql_stmt_init()`. To prepare the statement on the server, call `mysql_stmt_prepare()` and pass it a string containing the SQL statement.
2. If the statement will produce a result set, call `mysql_stmt_result_metadata()` to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.
3. Set the values of any parameters using `mysql_stmt_bind_param()`. All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.
4. Call `mysql_stmt_execute()` to execute the statement.
5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling `mysql_stmt_bind_result()`.
6. Fetch the data into the buffers row by row by calling `mysql_stmt_fetch()` repeatedly until no more rows are found.
7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When `mysql_stmt_prepare()` is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.
- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

- The client uses the statement handle and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and the number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the current row of the result set and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error number, error message, and SQLSTATE code using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

Prepared Statement Logging

For prepared statements that are executed with the `mysql_stmt_prepare()` and `mysql_stmt_execute()` C API functions, the server writes `Prepare` and `Execute` lines to the general query log so that you can tell when statements are prepared and executed.

Suppose that you prepare and execute a statement as follows:

1. Call `mysql_stmt_prepare()` to prepare the statement string `"SELECT ?"`.
2. Call `mysql_stmt_bind_param()` to bind the value `3` to the parameter in the prepared statement.
3. Call `mysql_stmt_execute()` to execute the prepared statement.

As a result of the preceding calls, the server writes the following lines to the general query log:

```
Prepare  [1] SELECT ?
Execute  [1] SELECT 3
```

Each `Prepare` and `Execute` line in the log is tagged with a `[N]` statement identifier so that you can keep track of which prepared statement is being logged. *`N`* is a positive integer. If there are multiple prepared statements active simultaneously for the client, *`N`* may be greater than 1. Each `Execute` lines shows a prepared statement after substitution of data values for `?` parameters.

Version notes: `Prepare` lines are displayed without `[N]` before MySQL 4.1.10. `Execute` lines are not displayed at all before MySQL 4.1.10.

17.6.10 C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions described in detail in the following sections.

Note

In MySQL 4.1.2, the names of several prepared statement functions were changed, as shown here:

Old Name	New Name
`mysql_bind_param()`	`mysql_stmt_bind_param()`
`mysql_bind_result()`	`mysql_stmt_bind_result()`
`mysql_prepare()`	`mysql_stmt_prepare()`
`mysql_execute()`	`mysql_stmt_execute()`
`mysql_fetch()`	`mysql_stmt_fetch()`
`mysql_fetch_column()`	`mysql_stmt_fetch_column()`
`mysql_param_count()`	`mysql_stmt_param_count()`
`mysql_param_result()`	`mysql_stmt_param_metadata()`
`mysql_get_metadata()`	`mysql_stmt_result_metadata()`
`mysql_send_long_data()`	`mysql_stmt_send_long_data()`

All functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

Also in 4.1.2, the signature of the `mysql_stmt_prepare()` function was changed to `int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)`. To create a `MYSQL_STMT` handle, you should use the `mysql_stmt_init()` function.

17.6.10.1 `mysql_stmt_affected_rows()`

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_affected_rows()` may be called immediately after executing a statement with `mysql_stmt_execute()`. It is like `mysql_affected_rows()` but for prepared statements. For a description of what the affected-rows value returned by this function means, See Section 17.6.6.1, “`mysql_affected_rows()`”.

This function was added in MySQL 4.1.0.

Errors

None.

Example

See the Example in Section 17.6.10.10, “`mysql_stmt_execute()`”.

17.6.10.2 `mysql_stmt_attr_get()`

```
my_bool mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The `option` argument is the option that you want to get; the `arg` should point to a variable that should contain the option value. If the option is an integer, `arg` should point to the value of the integer.

See Section 17.6.10.3, “`mysql_stmt_attr_set()`”, for a list of options and option types.

Note

In MySQL 4.1, `mysql_stmt_attr_get()` uses `unsigned long *`, not `my_bool *`, for `STMT_ATTR_UPDATE_MAX_LENGTH`. This is corrected in MySQL 5.1.7.

This function was added in MySQL 4.1.2.

Return Values

Zero if successful. Nonzero if `option` is unknown.

Errors

None.

17.6.10.3 `mysql_stmt_attr_set()`

```
my_bool mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
const void *arg)
```

Description

Can be used to affect behavior for a prepared statement. In MySQL 4.1, the `option` argument can take the single value `STMT_ATTR_UPDATE_MAX_LENGTH`; the `arg` argument is a pointer of type `my_bool *`. If `arg` points to the value `1`, then the metadata `MYSQL_FIELD->max_length` in `mysql_stmt_store_result()` is updated when the prepared statement is executed.

Note

In MySQL 4.1, `mysql_stmt_attr_get()` uses `unsigned int *`, not `my_bool *`, for `STMT_ATTR_UPDATE_MAX_LENGTH`. This is corrected in MySQL 5.1.7.

This function was added in MySQL 4.1.2. Additional options are planned for this function in later versions of MySQL.

Return Values

Zero if successful. Nonzero if `option` is unknown.

Errors

None.

17.6.10.4 mysql_stmt_bind_param()

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_param()` is used to bind input data for the parameter markers in the SQL statement that was passed to `mysql_stmt_prepare()`. It uses `MYSQL_BIND` structures to supply the data. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each `?` parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?,?,?)
```

When you bind the parameters, the array of `MYSQL_BIND` structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

Section 17.6.8, “C API Prepared Statement Data Structures”, describes the members of each `MYSQL_BIND` element and how they should be set to provide input values.

This function was added in MySQL 4.1.2.

Return Values

Zero if the bind operation was successful. Nonzero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE` [1590]

The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

See the Example in Section 17.6.10.10, "`mysql_stmt_execute()`".

17.6.10.5 `mysql_stmt_bind_result()`

```
my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_result()` is used to associate (that is, bind) output columns in the result set to data buffers and length buffers. When `mysql_stmt_fetch()` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling `mysql_stmt_fetch()`. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each column of the result set. If you do not bind columns to `MYSQL_BIND` structures, `mysql_stmt_fetch()` simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol does not return data values in chunks.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysql_stmt_fetch()` is called. Suppose that an application binds the columns in a result set and calls `mysql_stmt_fetch()`. The client/server protocol returns data in the bound buffers. Then suppose that the application binds the columns to a different set of buffers. The protocol places data into the newly bound buffers when the next call to `mysql_stmt_fetch()` occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and length of the output buffer into which the value should be stored. Section 17.6.8, "C API Prepared Statement Data Structures", describes the members of each `MYSQL_BIND` element and how they should be set to receive output values.

This function was added in MySQL 4.1.2.

Return Values

Zero if the bind operation was successful. Nonzero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE` [1590]

  The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

See the Example in Section 17.6.10.11, “`mysql_stmt_fetch()`”.

17.6.10.6 `mysql_stmt_close()`

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handle pointed to by `stmt`.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

This function was added in MySQL 4.1.0.

Return Values

Zero if the statement was freed successfully. Nonzero if an error occurred.

Errors

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

See the Example in Section 17.6.10.10, “`mysql_stmt_execute()`”.

17.6.10.7 `mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from `0` to `mysql_stmt_num_rows(stmt)-1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

This function was added in MySQL 4.1.1.

Return Values

None.

Errors

None.

17.6.10.8 mysql_stmt_errno()

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix B, *Errors, Error Codes, and Common Problems*.

This function was added in MySQL 4.1.0.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

17.6.10.9 mysql_stmt_error()

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string (`""`) is returned if no error occurred. Either of these two tests can be used to check for an error:

```
if(*mysql_stmt_errno(stmt))
{
  // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages.

This function was added in MySQL 4.1.0.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

17.6.10.10 mysql_stmt_execute()

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handle. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

Statement processing following `mysql_stmt_execute()` depends on the type of statement:

- For an `UPDATE`, `DELETE`, or `INSERT`, the number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`.
- For a statement such as `SELECT` that generates a result set, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to Section 17.6.10.11, “`mysql_stmt_fetch()`”.

  Do not following invocation of `mysql_stmt_execute()` with a call to `mysql_store_result()` or `mysql_use_result()`. Those functions are not intended for processing results from prepared statements.

For statements that generate a result set, you can request that `mysql_stmt_execute()` open a cursor for the statement by calling `mysql_stmt_attr_set()` before executing the statement. If you execute a statement multiple times, `mysql_stmt_execute()` closes any open cursor before opening a new one.

This function was added in MySQL 4.1.2.

Return Values

Zero if execution was successful. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using `mysql_stmt_init()`, `mysql_stmt_prepare()`, `mysql_stmt_param_count()`, `mysql_stmt_bind_param()`, `mysql_stmt_execute()`, and `mysql_stmt_affected_rows()`. The `mysql` variable is assumed to

be a valid connection handle. For an example that shows how to retrieve data, see Section 17.6.10.11, “mysql_stmt_fetch()”.

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                                     col2 VARCHAR(40),\
                                                     col3 SMALLINT,\
                                                     col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO \
                       test_table(col1,col2,col3) \
                       VALUES(?,?,?)"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[3];
my_ulonglong  affected_rows;
int           param_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
unsigned long str_length;
my_bool       is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
  fprintf(stderr, " DROP TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
  fprintf(stderr, " CREATE TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/*  sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Bind the data for all 3 parameters */
```

```
memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need
   to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_param() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Specify the data values for the first row */
int_data= 10;             /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string  */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
               (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Specify data values for second row,
   then re-execute the statement */
int_data= 1000;
strncpy(str_data, "
        The most popular Open Source database",
        STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;         /* smallint */
```

```
is_null= 0;              /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
               (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

Note

For complete examples on the use of prepared statement functions, refer to the file `tests/mysql_client_test.c`. This file can be obtained from a MySQL source distribution or from the Bazaar source repository.

17.6.10.11 `mysql_stmt_fetch()`

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_fetch()` returns the next row in the result set. It can be called only while the result set exists; that is, after a call to `mysql_stmt_execute()` for a statement such as `SELECT` that produces a result set.

`mysql_stmt_fetch()` returns row data using the buffers bound by `mysql_stmt_bind_result()`. It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the `length` pointer. All columns must be bound by the application before it calls `mysql_stmt_fetch()`.

By default, result sets are fetched unbuffered a row at a time from the server. To buffer the entire result set on the client, call `mysql_stmt_store_result()` after binding the data buffers and before caling `mysql_stmt_fetch()`.

If a fetched data value is a `NULL` value, the `*is_null` value of the corresponding `MYSQL_BIND` structure contains TRUE (1). Otherwise, the data and its length are returned in the `*buffer` and `*length` elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by `data_length`.

Type	Length
`MYSQL_TYPE_TINY`	1

Type	Length
MYSQL_TYPE_SHORT	2
MYSQL_TYPE_LONG	4
MYSQL_TYPE_LONGLONG	8
MYSQL_TYPE_FLOAT	4
MYSQL_TYPE_DOUBLE	8
MYSQL_TYPE_TIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATE	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATETIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_BLOB	data_length

This function was added in MySQL 4.1.2.

In some cases you might want to determine the length of a column value before fetching it with mysql_stmt_fetch(). For example, the value might be a long string or BLOB value for which you want to know how much space must be allocated. To accomplish this, you can use these strategies:

- Before invoking mysql_stmt_fetch() to retrieve individual rows, pass STMT_ATTR_UPDATE_MAX_LENGTH to mysql_stmt_attr_set(), then invoke mysql_stmt_store_result() to buffer the entire result on the client side. Setting the STMT_ATTR_UPDATE_MAX_LENGTH attribute causes the maximal length of column values to be indicated by the max_length member of the result set metadata returned by mysql_stmt_result_metadata().
- Invoke mysql_stmt_fetch() with a zero-length buffer for the column in question and a pointer in which the real length can be stored. Then use the real length with mysql_stmt_fetch_column().

```
real_length= 0;

bind[0].buffer= 0;
bind[0].buffer_length= 0;
bind[0].length= &real_length
mysql_stmt_bind_result(stmt, bind);

mysql_stmt_fetch(stmt);
if (real_length > 0)
{
  data= malloc(real_length);
  bind[0].buffer= data;
  bind[0].buffer_length= real_length;
  mysql_stmt_fetch_column(stmt, bind, 0, 0);
}
```

Return Values

Return Value	Description
0	Successful, the data has been fetched to application data buffers.
1	Error occurred. Error code and message can be obtained by calling mysql_stmt_errno() and mysql_stmt_error().
MYSQL_NO_DATA	No more rows/data exists
MYSQL_DATA_TRUNCATED	Data truncation occurred

`MYSQL_DATA_TRUNCATED` is not returned unless truncation reporting is enabled with `mysql_options()`. To determine which parameters were truncated when this value is returned, check the `error` members of the `MYSQL_BIND` parameter structures.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

- `CR_UNSUPPORTED_PARAM_TYPE` [1590]

  The buffer type is `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, but the data type is not `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP`.

- All other unsupported conversion errors are returned from `mysql_stmt_bind_result()`.

Example

The following example demonstrates how to fetch data from a table using `mysql_stmt_result_metadata()`, `mysql_stmt_bind_result()`, and `mysql_stmt_fetch()`. (This example expects to retrieve the two rows inserted by the example shown in Section 17.6.10.10, “`mysql_stmt_execute()`”.) The `mysql` variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 \
                       FROM test_table"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[4];
MYSQL_RES     *prepare_meta_result;
MYSQL_TIME    ts;
unsigned long length[4];
int           param_count, column_count, row_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
my_bool       is_null[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
```

```
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr,
         " mysql_stmt_result_metadata(), \
           returned no meta information\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout,
        " total columns in SELECT statement: %d\n",
        column_count);

if (column_count != 4) /* validate column count */
{
  fprintf(stderr, " invalid column count returned by MySQL\n");
  exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
```

```
/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Now buffer all results to client (optional step) */
if (mysql_stmt_store_result(stmt))
{
  fprintf(stderr, " mysql_stmt_store_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
  row_count++;
  fprintf(stdout, "  row %d\n", row_count);

  /* column 1 */
  fprintf(stdout, "   column1 (integer)  : ");
  if (is_null[0])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

  /* column 2 */
  fprintf(stdout, "   column2 (string)   : ");
  if (is_null[1])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

  /* column 3 */
  fprintf(stdout, "   column3 (smallint) : ");
  if (is_null[2])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

  /* column 4 */
  fprintf(stdout, "   column4 (timestamp): ");
  if (is_null[3])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                    ts.year, ts.month, ts.day,
                    ts.hour, ts.minute, ts.second,
                    length[3]);
  fprintf(stdout, "\n");
```

```
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
  fprintf(stderr, " MySQL failed to return all rows\n");
  exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);


/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

17.6.10.12 mysql_stmt_fetch_column()

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int
column, unsigned long offset)
```

Description

Fetch one column from the current result set row. `bind` provides the buffer where data should be placed. It should be set up the same way as for `mysql_stmt_bind_result()`. `column` indicates which column to fetch. The first column is numbered 0. `offset` is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

This function was added in MySQL 4.1.2.

Return Values

Zero if the value was fetched successfully. Nonzero if an error occurred.

Errors

- `CR_INVALID_PARAMETER_NO` [1590]

  Invalid column number.

- `CR_NO_DATA` [1591]

  The end of the result set has already been reached.

17.6.10.13 mysql_stmt_field_count()

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as `INSERT` or `DELETE` that do not produce result sets.

`mysql_stmt_field_count()` can be called after you have prepared a statement by invoking `mysql_stmt_prepare()`.

This function was added in MySQL 4.1.3.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

17.6.10.14 mysql_stmt_free_result()

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement. If there is a cursor open for the statement, `mysql_stmt_free_result()` closes it.

This function was added in MySQL 4.1.1.

Return Values

Zero if the result set was freed successfully. Nonzero if an error occurred.

Errors

17.6.10.15 mysql_stmt_init()

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Create a `MYSQL_STMT` handle. The handle should be freed with `mysql_stmt_close(MYSQL_STMT *)`.

This function was added in MySQL 4.1.2.

See also Section 17.6.8, “C API Prepared Statement Data Structures”, for more information.

Return Values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

17.6.10.16 mysql_stmt_insert_id()

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the prepared `INSERT` or `UPDATE` statement. Use this function after you have executed prepared `INSERT` statement into a table which contains an `AUTO_INCREMENT` field.

See Section 17.6.6.35, "`mysql_insert_id()`", for more information.

This function was added in MySQL 4.1.2.

Return Values

Value for `AUTO_INCREMENT` column which was automatically generated or explicitly set during execution of prepared statement, or value generated by `LAST_INSERT_ID(expr)` [874] function. Return value is undefined if statement does not set `AUTO_INCREMENT` value.

Errors

None.

17.6.10.17 `mysql_stmt_num_rows()`

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of `mysql_stmt_num_rows()` depends on whether you used `mysql_stmt_store_result()` to buffer the entire result set in the statement handle. If you use `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` may be called immediately. Otherwise, the row count is unavailable unless you count the rows as you fetch them.

`mysql_stmt_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_stmt_affected_rows()`.

This function was added in MySQL 4.1.1.

Return Values

The number of rows in the result set.

Errors

None.

17.6.10.18 `mysql_stmt_param_count()`

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

This function was added in MySQL 4.1.2.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

See the Example in Section 17.6.10.10, “`mysql_stmt_execute()`”.

17.6.10.19 `mysql_stmt_param_metadata()`

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

This function currently does nothing.

This function was added in MySQL 4.1.2.

Description

Return Values

Errors

17.6.10.20 `mysql_stmt_prepare()`

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *stmt_str, unsigned long
length)
```

Description

Given the statement handle returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `stmt_str` and returns a status value. The string length should be given by the `length` argument. The string must consist of a single SQL statement. You should not add a terminating semicolon (“`;`”) or `\g` to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark (`?`) characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are permitted in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value. However, they are not permitted for identifiers (such as table or column names), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

This function was added in MySQL 4.1.2.

Return Values

Zero if the statement was prepared successfully. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

  The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

If the prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns nonzero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

See the Example in Section 17.6.10.10, “`mysql_stmt_execute()`”.

17.6.10.21 `mysql_stmt_reset()`

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Resets a prepared statement on client and server to state after prepare. It resets the statement on the server, data sent using `mysql_stmt_send_long_data()`, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

This function was added in MySQL 4.1.1.

Return Values

Zero if the statement was reset successfully. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

17.6.10.22 `mysql_stmt_result_metadata()`

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to `mysql_stmt_prepare()` is one that produces a result set, `mysql_stmt_result_metadata()` returns the result set metadata in the form of a pointer to a `MYSQL_RES` structure that can be used to process the meta information such as number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysql_free_result()`. This is similar to the way you free a result set obtained from a call to `mysql_store_result()`.

The result set returned by `mysql_stmt_result_metadata()` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with `mysql_stmt_fetch()`.

This function was added in MySQL 4.1.2.

Return Values

A `MYSQL_RES` result structure. `NULL` if no meta information exists for the prepared query.

Errors

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

See the Example in Section 17.6.10.11, “mysql_stmt_fetch()”.

17.6.10.23 mysql_stmt_row_seek()

```
MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

This function was added in MySQL 4.1.1.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

17.6.10.24 mysql_stmt_row_tell()

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

This function was added in MySQL 4.1.1.

Return Values

The current offset of the row cursor.

Errors

None.

17.6.10.25 mysql_stmt_send_long_data()

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int
parameter_number, const char *data, unsigned long length)
```

Description

Enables an application to send parameter data to the server in pieces (or “chunks”). Call this function after `mysql_stmt_bind_param()` and before `mysql_stmt_execute()`. It can be called multiple times to

send the parts of a character or binary data value for a column, which must be one of the `TEXT` or `BLOB` data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

Note

The next `mysql_stmt_execute()` call ignores the bind buffer for all parameters that have been used with `mysql_stmt_send_long_data()` since last `mysql_stmt_execute()` or `mysql_stmt_reset()`.

If you want to reset/forget the sent data, you can do it with `mysql_stmt_reset()`. See Section 17.6.10.21, “`mysql_stmt_reset()`”.

This function was added in MySQL 4.1.2.

Return Values

Zero if the data is sent successfully to server. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

Example

The following example demonstrates how to send the data for a `TEXT` column in chunks. It inserts the data value `'MySQL - The most popular Open Source database'` into the `text_column` column. The `mysql` variable is assumed to be a valid connection handle.

```
#define INSERT_QUERY "INSERT INTO \
                      test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long       length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
```

```
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
 memset(bind, 0, sizeof(bind));
 bind[0].buffer_type= MYSQL_TYPE_STRING;
 bind[0].length= &length;
 bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, "\n param bind failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply data in chunks to server */
 if (mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply the next piece of data */
 if (mysql_stmt_send_long_data(stmt,0,
          " - The most popular Open Source database",40))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Now, execute the query */
 if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, "\n mysql_stmt_execute failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

17.6.10.26 `mysql_stmt_sqlstate()`

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. `"00000"` means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see Appendix B, *Errors, Error Codes, and Common Problems*.

Note that not all MySQL errors are yet mapped to SQLSTATE codes. The value `"HY000"` (general error) is used for unmapped errors.

This function was added to MySQL 4.1.1.

Return Values

A null-terminated character string containing the SQLSTATE error code.

17.6.10.27 mysql_stmt_store_result()

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

Result sets are produced by calling `mysql_stmt_execute()` to executed prepared statements for SQL statements such as `SELECT`, `SHOW`, `DESCRIBE`, and `EXPLAIN`. By default, result sets for successfully executed prepared statements are not buffered on the client and `mysql_stmt_fetch()` fetches them one at a time from the server. To cause the complete result set to be buffered on the client, call `mysql_stmt_store_result()` after binding data buffers with `mysql_stmt_bind_result()` and before calling `mysql_stmt_fetch()` to fetch rows. (For an example, see Section 17.6.10.11, “mysql_stmt_fetch()”.)

`mysql_stmt_store_result()` is optional for result set processing, unless you will call `mysql_stmt_data_seek()`, `mysql_stmt_row_seek()`, or `mysql_stmt_row_tell()`. Those functions require a seekable result set.

It is unnecessary to call `mysql_stmt_store_result()` after executing an SQL statement that does not produce a result set, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to Section 17.6.10.22, “mysql_stmt_result_metadata()”.

Note

MySQL does not by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications do not need `max_length`. If you want `max_length` to be updated, you can call `mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See Section 17.6.10.3, “mysql_stmt_attr_set()”.

This function was added in MySQL 4.1.0.

Return Values

Zero if the results are buffered successfully. Nonzero if an error occurred.

Errors

- `CR_INVALID_BUFFER_USE` [1590]

  The parameter does not have a string or binary type.

- `CR_COMMANDS_OUT_OF_SYNC` [1588]

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY` [1588]

  Out of memory.

- `CR_SERVER_GONE_ERROR` [1588]

  The MySQL server has gone away.

- `CR_SERVER_LOST` [1588]

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR` [1587]

  An unknown error occurred.

If the application is linked to the embedded server library, runtime error messages will indicate the `libmysqld` rather than `libmysqlclient` library, but the solution to the problem is the same as just described.

17.6.11 C API Threaded Function Descriptions

To create a threaded client, use the functions described in the following sections. See also Section 17.6.3.2, “Writing C API Threaded Client Programs”.

17.6.11.1 `my_init()`

```
void my_init(void)
```

Description

`my_init()` initializes some global variables that MySQL needs. If you are using a thread-safe client library, it also calls `mysql_thread_init()` for this thread.

It is necessary for `my_init()` to be called early in the initialization phase of a program's use of the MySQL library. However, `my_init()` is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you ensure that your program invokes one of those functions before any other MySQL calls, there is no need to invoke `my_init()` explicitly.

To access the prototype for `my_init()`, your program should include these header files:

```
#include <my_global.h>
#include <my_sys.h>
```

Return Values

None.

17.6.11.2 `mysql_thread_end()`

```
void mysql_thread_end(void)
```

Description

This function needs to be called before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

`mysql_thread_end()` *is not invoked automatically by the client library*. It must be called explicitly to avoid a memory leak.

Return Values

None.

17.6.11.3 `mysql_thread_init()`

```
my_bool mysql_thread_init(void)
```

Description

This function must be called early within each created thread to initialize thread-specific variables. However, you may not necessarily need to invoke it explicitly: `mysql_thread_init()` is automatically called by `my_init()`, which itself is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you invoke any of those functions, `mysql_thread_init()` will be called for you.

Return Values

Zero for success. Nonzero if an error occurred.

17.6.11.4 `mysql_thread_safe()`

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client library is compiled as thread-safe.

Return Values

1 if the client library is thread-safe, 0 otherwise.

17.6.12 C API Embedded Server Function Descriptions

MySQL applications can be written to use an embedded server. See Section 17.5, “libmysqld, the Embedded MySQL Server Library”. To write such an application, you must link it against the `libmysqld` library by using the `-lmysqld` flag rather than linking it against the `libmysqlclient` client library by using the `-lmysqlclient` flag. However, the calls to initialize and finalize the library are the same whether you write a client application or one that uses the embedded server: Call `mysql_library_init()` to initialize the library and `mysql_library_end()` when you are done with it. See Section 17.6.5, “C API Function Overview”.

`mysql_library_init()` and `mysql_library_end()` are available as of MySQL 4.1.10. For earlier versions of MySQL 4.1, call `mysql_server_init()` and `mysql_server_end()` instead, which are equivalent. `mysql_library_init()` and `mysql_library_end()` actually are `#define` symbols that make them equivalent to `mysql_server_init()` and `mysql_server_end()`, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`.

17.6.12.1 `mysql_server_init()`

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function initializes the MySQL library, which must be done before you call any other MySQL function.

As of MySQL 4.1.10, `mysql_server_init()` is deprecated and you should call `mysql_library_init()` instead. See Section 17.6.6.38, “`mysql_library_init()`”.

Return Values

Zero for success. Nonzero if an error occurred.

17.6.12.2 mysql_server_end()

```
void mysql_server_end(void)
```

Description

This function finalizes the MySQL library. You should call it when you are done using the library.

As of MySQL 4.1.10, mysql_server_end() is deprecated and you should call mysql_library_end() instead. See Section 17.6.6.37, “mysql_library_end()”.

Return Values

None.

17.6.13 Common Questions and Problems When Using the C API

17.6.13.1 Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success

It is possible for mysql_store_result() to return NULL following a successful call to mysql_query(). When this happens, it means one of the following conditions occurred:

- There was a malloc() failure (for example, if the result set was too large).
- The data couldn't be read (an error occurred on the connection).
- The query returned no data (for example, it was an INSERT, UPDATE, or DELETE).

You can always check whether the statement should have produced a nonempty result by calling mysql_field_count(). If mysql_field_count() returns zero, the result is empty and the last query was a statement that does not return values (for example, an INSERT or a DELETE). If mysql_field_count() returns a nonzero value, the statement should have produced a nonempty result. See the description of the mysql_field_count() function for an example.

You can test for an error by calling mysql_error() or mysql_errno().

17.6.13.2 What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- mysql_affected_rows() returns the number of rows affected by the last query when doing an INSERT, UPDATE, or DELETE.

  In MySQL 3.23, there is an exception when DELETE is used without a WHERE clause. In this case, the table is re-created as an empty table and mysql_affected_rows() returns zero for the number of records affected. In MySQL 4.0 and later, DELETE always returns the correct number of rows deleted. For a fast re-create, use TRUNCATE TABLE.

- mysql_num_rows() returns the number of rows in a result set. With mysql_store_result(), mysql_num_rows() may be called as soon as mysql_store_result() returns. With mysql_use_result(), mysql_num_rows() may be called only after you have fetched all the rows with mysql_fetch_row().

- mysql_insert_id() returns the ID generated by the last query that inserted a row into a table with an AUTO_INCREMENT index. See Section 17.6.6.35, “mysql_insert_id()”.

- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

17.6.13.3 How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

You can check from your C applications whether a value was stored in an `AUTO_INCREMENT` column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an `INSERT` with an `AUTO_INCREMENT` index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

When a new `AUTO_INCREMENT` value has been generated, you can also obtain it by executing a `SELECT LAST_INSERT_ID()` statement with `mysql_query()` and retrieving the value from the result set returned by the statement.

For `LAST_INSERT_ID()` [874], the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another `AUTO_INCREMENT` column with a nonmagic value (that is, a value that is not `NULL` and not `0`). Using `LAST_INSERT_ID()` [874] and `AUTO_INCREMENT` columns simultaneously from multiple clients is perfectly valid. Each client will receive the last inserted ID for the last statement *that* client executed.

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');         # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

Note that `mysql_insert_id()` returns the value stored into an `AUTO_INCREMENT` column, whether that value is automatically generated by storing `NULL` or `0` or was specified as an explicit value. `LAST_INSERT_ID()` [874] returns only automatically generated `AUTO_INCREMENT` values. If you store an explicit value other than `NULL` or `0`, it does not affect the value returned by `LAST_INSERT_ID()` [874].

For more information on obtaining the last ID in an `AUTO_INCREMENT` column:

- For information on `LAST_INSERT_ID()` [874], which can be used within an SQL statement, see Section 11.13, “Information Functions”.
- For information on `mysql_insert_id()`, the function you use from within the C API, see Section 17.6.6.35, “`mysql_insert_id()`”.
- For information on obtaining the auto-incremented value when using Connector/J see MySQL Connector/J Developer Guide.
- For information on obtaining the auto-incremented value when using Connector/ODBC see Obtaining Auto-Increment Values.

17.6.14 Controlling Automatic Reconnection Behavior

The MySQL client library can perform an automatic reconnection to the server if it finds that the connection is down when you attempt to send a statement to the server to be executed. If auto-reconnect is enabled, the library tries once to reconnect to the server and send the statement again.

If the connection has gone down, the `mysql_ping()` function performs a reconnect if auto-reconnect is enabled. If auto-reconnect is disabled, `mysql_ping()` returns an error instead.

Some client programs might provide the capability of controlling automatic reconnection. For example, `mysql` reconnects by default, but the `--skip-reconnect` [264] option can be used to suppress this behavior.

If an automatic reconnection does occur (for example, as a result of calling `mysql_ping()`), there is no explicit indication of it. To check for reconnection, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier changed.

Automatic reconnection can be convenient because you need not implement your own reconnect code, but if a reconnection does occur, several aspects of the connection state are reset and your application will not know about it. The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session system variables are reinitialized to the values of the corresponding global system variables, including system variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.
- `HANDLER` variables are closed.
- The value of `LAST_INSERT_ID()` [874] is reset to 0.
- Locks acquired with `GET_LOCK()` [877] are released.

If the connection drops, it is possible that the session associated with the connection on the server side will still be running if the server has not yet detected that the client is no longer connected. In this case, any locks held by the original connection still belong to that session, so you may want to kill it by calling `mysql_kill()`.

17.6.15 C API Support for Multiple Statement Execution

By default, `mysql_query()` and `mysql_real_query()` interpret their statement string argument as a single statement to be executed, and you process the result according to whether the statement produces a result set (a set of rows, as for `SELECT`) or an affected-rows count (as for `INSERT`, `UPDATE`, and so forth).

MySQL 4.1 also supports the execution of a string containing multiple statements separated by semicolon (“`;`”) characters. This capability is enabled by special options that are specified either when you connect to the server with `mysql_real_connect()` or after connecting by calling` `mysql_set_server_option()`.

Executing a multiple-statement string can produce multiple result sets or row-count indicators. Processing these results involves a different approach than for the single-statement case: After handling the result from the first statement, it is necessary to check whether more results exist and process them in turn if so. To support multiple-result processing, the C API includes the `mysql_more_results()` and `mysql_next_result()` functions. These functions are used at the end of a loop that iterates as long as more results are available. *Failure to process the result this way may result in a dropped connection to the server.*

The multiple statement and result capabilities can be used only with `mysql_query()` or `mysql_real_query()`. They cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement. See Section 17.6.7, “C API Prepared Statements”.

To enable multiple-statement execution and result processing, the following options may be used:

- The `mysql_real_connect()` function has a `flags` argument for which two option values are relevent:
  - `CLIENT_MULTI_RESULTS` enables the client program to process multiple results.
  - `CLIENT_MULTI_STATEMENTS` enables `mysql_query()` and `mysql_real_query()` to execute statement strings containing multiple statements separated by semicolons. This option also enables `CLIENT_MULTI_RESULTS` implicitly, so a `flags` argument of `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()` is equivalent to an argument of `CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS`. That is, `CLIENT_MULTI_STATEMENTS` is sufficient to enable multiple-statement execution and all multiple-result processing.
- After the connection to the server has been established, you can use the `mysql_set_server_option()` function to enable or disable multiple-statement execution by passing it an argument of `MYSQL_OPTION_MULTI_STATEMENTS_ON` or `MYSQL_OPTION_MULTI_STATEMENTS_OFF`.

The following procedure outlines a suggested strategy for handling multiple statements:

1. Pass `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()`, to fully enable multiple-statement execution and multiple-result processing.
2. After calling `mysql_query()` or `mysql_real_query()` and verifying that it succeeds, enter a loop within which you process statement results.
3. For each iteration of the loop, handle the current statement result, retrieving either a result set or an affected-rows count. If an error occurs, exit the loop.
4. At the end of the loop, call `mysql_next_result()` to check whether another result exists and initiate retrieval for it if so. If no more results are available, exit the loop.

One possible implementation of the preceding strategy is shown following. The final part of the loop can be reduced to a simple test of whether `mysql_next_result()` returns nonzero. The code as written distinguishes between no more results and an error, which enables a message to be printed for the latter occurrence.

```
/* connect to server with the CLIENT_MULTI_STATEMENTS option */
if (mysql_real_connect (mysql, host_name, user_name, password,
    db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
{
  printf("mysql_real_connect() failed\n");
  mysql_close(mysql);
```

```
  exit(1);
}

/* execute multiple statements */
status = mysql_query(mysql,
                     "DROP TABLE IF EXISTS test_table;\
                      CREATE TABLE test_table(id INT);\
                      INSERT INTO test_table VALUES(10);\
                      UPDATE test_table SET id=20 WHERE id=10;\
                      SELECT * FROM test_table;\
                      DROP TABLE test_table");
if (status)
{
  printf("Could not execute statement(s)");
  mysql_close(mysql);
  exit(0);
}

/* process each statement result */
do {
  /* did current statement return data? */
  result = mysql_store_result(mysql);
  if (result)
  {
    /* yes; process rows and free the result set */
    process_result_set(mysql, result);
    mysql_free_result(result);
  }
  else          /* no result set or error */
  {
    if (mysql_field_count(mysql) == 0)
    {
      printf("%lld rows affected\n",
             mysql_affected_rows(mysql));
    }
    else  /* some error occurred */
    {
      printf("Could not retrieve result set\n");
      break;
    }
  }
  /* more results? -1 = no, >0 = error, 0 = yes (keep looping) */
  if ((status = mysql_next_result(mysql)) > 0)
    printf("Could not execute statement\n");
} while (status == 0);

mysql_close(mysql);
```

17.6.16 C API Prepared Statement Problems

Here follows a list of the currently known problems with prepared statements:

- `TIME`, `TIMESTAMP`, and `DATETIME` do not support parts of seconds (for example, from `DATE_FORMAT()` [832]).
- When converting an integer to string, `ZEROFILL` is honored with prepared statements in some cases where the MySQL server does not print the leading zeros. (For example, with `MIN(`*`number-with-zerofill`*`)` [884]).
- When converting a floating-point number to a string in the client, the rightmost digits of the converted value may differ slightly from those of the original value.
- *Prepared statements do not use the query cache, even in cases where a query does not contain any placeholders*. See Section 7.5.3.1, “How the Query Cache Operates”.

17.6.17 C API Prepared Statement Handling of Date and Time Values

The binary (prepared statement) protocol available in MySQL 4.1 and above enables you to send and receive date and time values (`DATE`, `TIME`, `DATETIME`, and `TIMESTAMP`), using the `MYSQL_TIME` structure. The members of this structure are described in Section 17.6.8, “C API Prepared Statement Data Structures”.

To send temporal data values, create a prepared statement using `mysql_stmt_prepare()`. Then, before calling `mysql_stmt_execute()` to execute the statement, use the following procedure to set up each temporal parameter:

1. In the `MYSQL_BIND` structure associated with the data value, set the `buffer_type` member to the type that indicates what kind of temporal value you're sending. For `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` values, set `buffer_type` to `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, respectively.

2. Set the `buffer` member of the `MYSQL_BIND` structure to the address of the `MYSQL_TIME` structure in which you pass the temporal value.

3. Fill in the members of the `MYSQL_TIME` structure that are appropriate for the type of temporal value to be passed.

Use `mysql_stmt_bind_param()` to bind the parameter data to the statement. Then you can call `mysql_stmt_execute()`.

To retrieve temporal values, the procedure is similar, except that you set the `buffer_type` member to the type of value you expect to receive, and the `buffer` member to the address of a `MYSQL_TIME` structure into which the returned value should be placed. Use `mysql_stmt_bind_result()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

Here is a simple example that inserts `DATE`, `TIME`, and `TIMESTAMP` data. The `mysql` variable is assumed to be a valid connection handle.

```
MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field, \
                           timestamp_field) VALUES(?,?,?");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
```

```
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..
```

17.7 MySQL PHP API

The MySQL PHP API manual is now published in standalone form, not as part of the MySQL Reference Manual. See MySQL and PHP.

17.8 MySQL Perl API

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI with MySQL, install the following:

1. The `DBI` module.
2. The `DBD::mysql` module. This is the DataBase Driver (DBD) module for Perl.
3. Optionally, the DBD module for any other type of database server you want to access.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

These sections contain information about using Perl with MySQL and writing MySQL applications in Perl:

- For installation instructions for Perl DBI support, see Section 2.14, “Perl Installation Notes”.
- For an example of reading options from option files, see Section 5.7.3, “Using Client Programs in a Multiple-Server Environment”.
- For secure coding tips, see Section 5.4.1, “General Security Guidelines”.
- For debugging tips, see Section 18.4.1.3, “Debugging `mysqld` under `gdb`”.
- For some Perl-specific environment variables, see Section 2.13, “Environment Variables”.
- For considerations for running on Mac OS X, see Using the Bundled MySQL on Mac OS X Server.
- For ways to quote string literals, see Section 8.1.1, “String Literals”.

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

```
shell> perldoc DBI
shell> perldoc DBI::FAQ
shell> perldoc DBD::mysql
```

You can also use `pod2man`, `pod2html`, and so on to translate this information into other formats.

- For online information about Perl DBI, visit the DBI Web site, http://dbi.perl.org/. That site hosts a general DBI mailing list. Oracle Corporation hosts a list specifically about `DBD::mysql`; see Section 1.7.1, “MySQL Mailing Lists”.

- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI Web site, http://dbi.perl.org/.

  For information that focuses specifically on using DBI with MySQL, see *MySQL and Perl for the Web* (Paul DuBois, New Riders, 2001). This book's Web site is http://www.kitebird.com/mysql-perl/.

17.9 MySQL Python API

`MySQLdb` is a third-party driver that provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at http://sourceforge.net/projects/mysql-python/.

The new MySQL Connector/Python component provides an interface to the same Python API, and is built into the MySQL Server and supported by Oracle. See MySQL Connector/Python Developer Guide for details on the Connector, as well as coding guidelines for Python applications and sample Python code.

17.10 MySQL Ruby APIs

Two APIs are available for Ruby programmers developing MySQL applications:

- The MySQL/Ruby API is based on the `libmysqlclient` API library. For information on installing and using the MySQL/Ruby API, see Section 17.10.1, “The MySQL/Ruby API”.

- The Ruby/MySQL API is written to use the native MySQL network protocol (a native driver). For information on installing and using the Ruby/MySQL API, see Section 17.10.2, “The Ruby/MySQL API”.

For background and syntax information about the Ruby language, see Ruby Programming Language.

17.10.1 The MySQL/Ruby API

The MySQL/Ruby module provides access to MySQL databases using Ruby through `libmysqlclient`.

For information on installing the module, and the functions exposed, see MySQL/Ruby.

17.10.2 The Ruby/MySQL API

The Ruby/MySQL module provides access to MySQL databases using Ruby through a native driver interface using the MySQL network protocol.

For information on installing the module, and the functions exposed, see Ruby/MySQL.

17.11 MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at http://www.xdobry.de/mysqltcl/.

17.12 MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at http://efsa.sourceforge.net/archive/ravits/mysql.htm.

Chapter 18 Extending MySQL

Table of Contents

18.1 MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. To track or contribute to MySQL development, follow the instructions in Section 2.9.2, “Installing MySQL from a Development Source Tree”. If you are interested in MySQL internals, you should also subscribe to our `internals` mailing list. This list has relatively low traffic. For details on how to subscribe, please see Section 1.7.1, “MySQL Mailing Lists”. Many MySQL developers at Oracle Corporation are on the `internals` list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

18.1.1 MySQL Threads

The MySQL server creates the following threads:

- Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

- Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

  For information about tuning the parameters that control thread resources, see Section 7.8.3, “How MySQL Uses Threads for Client Connections”.

- On a master replication server, connections from slave servers are handled like client connections: There is one thread per connected slave.

- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.
- A signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.
- If `InnoDB` is used, there will be 4 additional threads by default. Those are file I/O threads, controlled by the `innodb_file_io_threads` [1068] parameter. See Section 13.2.4, "`InnoDB` Startup Options and System Variables".
- If `mysqld` is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if you want to use the `thr_alarm()` code in your application without a dedicated signal handling thread.
- If the server is started with the `--flush_time=val` [411] option, a dedicated thread is created to flush all tables every *`val`* seconds.
- Each table for which `INSERT DELAYED` statements are issued gets its own thread. See Section 12.2.4.2, "`INSERT DELAYED` Syntax".

`mysqladmin processlist` only shows the connection, `INSERT DELAYED`, and replication threads.

18.1.2 The MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

You can also write your own test cases. For information about the MySQL Test Framework, including system requirements, see the manual available at http://dev.mysql.com/doc/mysqltest/2.0/en/.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (`mysqltest`), a Perl script to run all tests (`mysql-test-run.pl`), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type `make test` from the source root directory, or change location to the `mysql-test` directory and type `./mysql-test-run.pl`. If you have installed a binary distribution, change location to the `mysql-test` directory under the installation root directory (for example, `/usr/local/mysql/mysql-test`), and run `./mysql-test-run.pl`. All tests should succeed. If any do not, feel free to try to find out why and report the problem if it indicates a bug in MySQL. See Section 1.8, "How to Report Bugs or Problems".

If one test fails, you should run `mysql-test-run.pl` with the `--force` option to check whether any other tests fail.

If you have a copy of `mysqld` running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports `9306` or `9307`. If either of those ports is taken, you should set the `MTR_BUILD_THREAD` environment variable to an appropriate value, and the test suite will use a different set of ports for master, slave, NDB, and Instance Manager). For example:

```
shell> export MTR_BUILD_THREAD=31
```

```
shell> ./mysql-test-run.pl [options] [test_name]
```

In the `mysql-test` directory, you can run an individual test case with `./mysql-test-run.pl test_name`.

If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL `internals` mailing list. See Section 1.7.1, “MySQL Mailing Lists”.

Before MySQL 4.1, the `mysql-test-run` shell script is used instead of the `mysql-test-run.pl` Perl script. `mysql-test-run` does not try to run its own server by default but tries to use your currently running server. To override this and cause `mysql-test-run` to start its own server, run it with the `--local` option.

18.2 Adding New Functions to MySQL

There are two ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See Section 12.4.3.1, “`CREATE FUNCTION` Syntax for User-Defined Functions”.
- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the `mysqld` server and become available on a permanent basis.

Each method of creating compiled functions has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.
- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. (An incompatible change occurred in MySQL 4.1.1 for aggregate functions. A function named `xxx_clear()` must be defined rather than `xxx_reset()`.) For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as `ABS()` [817] or `SOUNDEX()` [801].

See Section 8.2.3, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the `sql/udf_example.cc` file that is provided in MySQL source distributions.

18.2.1 Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values and can accept arguments of those same types.

- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number and types of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns `NULL` or that an error occurred.

18.2.2 Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. MySQL source distributions include a file `sql/udf_example.cc` that defines five UDF functions. Consult this file to see how UDF calling conventions work. The `include/mysql_com.h` header file defines UDF-related symbols and data structures, although you need not include this header file directly; it is included by `mysql.h`.

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. Note that these constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see Section 2.9.3, “MySQL Source-Configuration Options”, and Section 2.9.4, “Dealing with Problems Compiling MySQL”.

To be able to use UDFs, you must link `mysqld` dynamically. Don't configure MySQL using `--with-mysqld-ldflags=-all-static`. If you want to use a UDF that needs to access symbols from `mysqld` (for example, the `metaphone` function in `sql/udf_example.cc` uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`). If you plan to use UDFs, the rule of thumb is to configure MySQL with `--with-mysqld-ldflags=-rdynamic` unless you have a very good reason not to.

If you must use a precompiled distribution of MySQL, use MySQL-Max, which contains a dynamically linked server that supports dynamic loading.

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the following discussion, the name “xxx” is used for an example function name. To distinguish between SQL and C/C++ usage, `XXX()` (uppercase) indicates an SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

Note

When using C++ you can encapsulate your C functions within:

```
extern "C" { ... }
```

This ensures that your C++ function names remain readable in the completed UDF.

The following list describes the C/C++ functions that you write to implement the interface for a function named `XXX()`. The main function, `xxx()`, is required. In addition, a UDF requires at least one of the other functions described here, for reasons discussed in Section 18.2.2.6, “User-Defined Function Security Precautions”.

- `xxx()`

  The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here.

SQL Type	C/C++ Type
STRING	char *
INTEGER	long long
REAL	double

- xxx_init()

  The initialization function for xxx(). If present, it can be used for the following purposes:

  - To check the number of arguments to XXX().
  - To verify that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the required types when the main function is called.
  - To allocate any memory required by the main function.
  - To specify the maximum length of the result.
  - To specify (for REAL functions) the maximum number of decimal places in the result.
  - To specify whether the result can be NULL.

- xxx_deinit()

  The deinitialization function for xxx(). If present, it should deallocate any memory allocated by the initialization function.

When an SQL statement invokes XXX(), MySQL calls the initialization function xxx_init() to let it perform any required setup, such as argument checking or memory allocation. If xxx_init() returns an error, MySQL aborts the SQL statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function xxx() once for each row. After all rows have been processed, MySQL calls the deinitialization function xxx_deinit() so that it can perform any required cleanup.

For aggregate functions that work like SUM() [884], you must also provide the following functions:

- xxx_reset() (required before 4.1.1)

  Reset the current aggregate value and insert the argument as the initial aggregate value for a new group.

- xxx_clear() (required starting from 4.1.1)

  Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- xxx_add()

  Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call xxx_init() to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the GROUP BY expression.

3. Before MySQL 4.1.1, call `xxx_clear()` for the first row in each new group. As of 4.1.1, call `xxx_clear()` for the first row in each new group.

4. Before MySQL 4.1.1, call `xxx_add()` for each new row that belongs in the same group, except for the first row. As of 4.1.1, call `xxx_add()` for each new row that belongs in the same group, including the first row.

5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.

6. Repeat steps 3 to 5 until all rows has been processed

7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not permitted to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

18.2.2.1 UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. Section 18.2.2, “Adding a New User-Defined Function”, describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- `my_bool maybe_null`

  `xxx_init()` should set `maybe_null` to `1` if `xxx()` can return `NULL`. The default value is `1` if any of the arguments are declared `maybe_null`.

- `unsigned int decimals`

  The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. For example, if the function is passed `1.34`, `1.345`, and `1.3`, the default would be 3, because `1.345` has 3 decimal digits.

  For arguments that have no fixed number of decimals, the `decimals` value is set to 31, which is 1 more than the maximum number of decimals permitted for the `DECIMAL`, `FLOAT`, and `DOUBLE` data types.

  A `decimals` value of 31 is used for arguments in cases such as a `FLOAT` or `DOUBLE` column declared without an explicit number of decimals (for example, `FLOAT` rather than `FLOAT(10,3)`) and for floating-point constants such as `1345E-3`. It is also used for string and other nonnumber arguments that might be converted within the function to numeric form.

  The value to which the `decimals` member is initialized is only a default. It can be changed within the function to reflect the actual calculation performed. The default is determined such that the largest number of decimals of the arguments is used. If the number of decimals is 31 for even one of the arguments, that is the value used for `decimals`.

- `unsigned int max_length`

  The maximum length of the result. The default `max_length` value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

  If you want to return a blob value, you can set `max_length` to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- `char *ptr`

  A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory among themselves. `xxx_init()` should allocate the memory and assign it to this pointer:

  ```
  initid->ptr = allocated_memory;
  ```

  In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

18.2.2.2 UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. Section 18.2.2, “Adding a New User-Defined Function”, describes the order in which MySQL calls these functions.

- `xxx_reset()`

  This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given `UDF_ARGS` argument as the first value in your internal summary value for the group. Declare `xxx_reset()` as follows:

```
void xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
               char *is_null, char *error);
```

`xxx_reset()` is needed only before MySQL 4.1.1. It is *not* needed or used as of MySQL 4.1.1, when the UDF interface changed to use `xxx_clear()` instead. However, you can define both `xxx_reset()` and `xxx_clear()` if you want to have your UDF work both before and after the interface change. (If you do include both functions, the `xxx_reset()` function in many cases can be implemented internally by calling `xxx_clear()` to reset all variables, and then calling `xxx_add()` to add the `UDF_ARGS` argument as the first value in the group.)

- `xxx_clear()`

  This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare `xxx_clear()` as follows:

  ```
  void xxx_clear(UDF_INIT *initid, char *is_null, char *error);
  ```

  `is_null` is set to point to `CHAR(0)` before calling `xxx_clear()`.

  If something went wrong, you can store a value in the variable to which the `error` argument points. `error` points to a single-byte variable, not to a string buffer.

  `xxx_clear()` is required only by MySQL 4.1.1 and above. Before MySQL 4.1.1, use `xxx_reset()` instead.

- `xxx_add()`

  This function is called for all rows that belong to the same group, with the exception that it is not called for the first row before MySQL 4.1.1 (see the preceding descriptions for the `xxx_clear()` and `xxx_reset()` functions). You should use it to add the value in the `UDF_ARGS` argument to your internal summary variable.

  ```
  void xxx_add(UDF_INIT *initid, UDF_ARGS *args,
               char *is_null, char *error);
  ```

The `xxx()` function for an aggregate UDF should be declared the same way as for a nonaggregate UDF. See Section 18.2.2.1, “UDF Calling Sequences for Simple Functions”.

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a nonaggregate UDF. See Section 18.2.2.4, “UDF Return Values and Error Handling”.

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for nonaggregate UDFs. See Section 18.2.2.3, “UDF Argument Processing”.

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error`! `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

18.2.2.3 UDF Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

  The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

  For other `UDF_ARGS` member values that are arrays, array references are zero-based. That is, refer to array members using index values from 0 to `args->arg_count` – 1.

- `enum Item_result *arg_type`

  A pointer to an array containing the types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, and `REAL_RESULT`.

  To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

  As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

- `char **args`

  `args->args` communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a nonconstant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)` [824]. A nonconstant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with nonconstant arguments.

  For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

If argument `i` represents `NULL`, `args->args[i]` is a null pointer (0). If the argument is not `NULL`, functions can refer to it as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to enable handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. Do not assume that the string is null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

```
double    real_val;
real_val = *((double*) args->args[i]);
```

- `unsigned long *lengths`

  For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

- `char *maybe_null`

  For the initialization function, the `maybe_null` array indicates for each argument whether the argument value might be null (0 if no, 1 if yes).

18.2.2.4 UDF Return Values and Error Handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message is returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string function should return a pointer to the result and set `*length` to the length (in bytes) of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

MySQL passes a buffer to the `xxx()` function using the `result` parameter. This buffer is sufficiently long to hold 255 characters, which as of MySQL 4.1 can be multi-byte characters. The `xxx()` function can store the result in this buffer if it fits, in which case the return value should be a pointer to the buffer. If the function stores the result in a different buffer, it should return a pointer to that buffer.

If your string function does not use the supplied buffer (for example, if it needs to return a string longer than 255 characters), you must allocate the space for your own buffer with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See Section 18.2.2.1, “UDF Calling Sequences for Simple Functions”.

To indicate a return value of `NULL` in the main function, set `*is_null` to `1`:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to `1`:

```
*error = 1;
```

If `xxx()` sets `*error` to `1` for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.)

Note

Before MySQL 3.22.10, you should set both `*error` and `*is_null`:

```
*error = 1;
*is_null = 1;
```

18.2.2.5 Compiling and Installing User-Defined Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.cc` that is included in the MySQL source distribution.

If a UDF will be referred to in statements that will be replicated to slave servers, you must ensure that every slave also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.cc` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it is more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP address for a host name.
- `reverse_lookup()` returns the host name for an IP address. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.
- `avgcost()` returns an average cost. This is an aggregate function.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.cc
```

If you are using `gcc`, you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example.so
```

You can easily determine the correct compiler options for your system by running this command in the `sql` directory of your MySQL source tree:

```
shell> make udf_example.o
```

You should run a compile command similar to the one that `make` displays, except that you should remove the `-c` option near the end of the line and add `-o udf_example.so` to the end of the line. (On some systems, you may need to leave the `-c` on the command.)

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.cc` produces a file named something like `udf_example.so` (the exact name may vary from platform to platform).

As of MySQL 4.1.25, copy the shared object to server's plugin directory and name it `udf_example.so`. This directory is given by the value of the `plugin_dir` [423] system variable.

Prior to MySQL 4.1.25, or if the value of `plugin_dir` [423] is empty, the shared object should be placed in a directory such as `/usr/lib` that is searched by your system's dynamic (runtime) linker, or you can add the directory in which you place the shared object to the linker configuration file (for example, `/etc/ld.so.conf`).

On many systems, you can also set the `LD_LIBRARY` or `LD_LIBRARY_PATH` environment variable to point at the directory where you have the files for your UDF. You should set the variable in `mysql.server` or `mysqld_safe` startup scripts and restart `mysqld`. You might do this if you want to place the object file in a directory accessible only to the server and not in a public directory. The `dlopen` manual page tells you which variable to use on your system.

The dynamic linker name is system-specific (for example, `ld-elf.so.1` on FreeBSD, `ld.so` on Linux, or `dyld` on Mac OS X). Consult your system documentation for information about the linker name and how to configure it.

On some systems, the `ldconfig` program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. Obtain the development source for MySQL 4.1. See Section 2.1.3, “How to Get MySQL”.
2. In the source repository, look in the `VC++Files/examples/udf_example` directory. There are files named `udf_example.def`, `udf_example.dsp`, and `udf_example.dsw` there.
3. In the source tree, look in the `sql` directory. Copy the `udf_example.cc` from this directory to the `VC++Files/examples/udf_example` directory and rename the file to `udf_example.cpp`.
4. Open the `udf_example.dsw` file with Visual Studio VC++ and use it to compile the UDFs as a normal project.

After the shared object file has been installed, notify `mysqld` about the new functions with the following statements. If object files have a suffix different from `.so` on your system, substitute the correct suffix throughout (for example, `.dll` on Windows).

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION sequence RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION reverse_lookup
    ->        RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
    ->        RETURNS REAL SONAME 'udf_example.so';
```

To delete functions, use `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION sequence;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the `func` system table in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the `INSERT` [492] or `DELETE` [491] privilege for the `mysql` database to create or drop functions, respectively.

You should not use `CREATE FUNCTION` to add a function that has previously been created. If you need to reinstall a function, you should remove it with `DROP FUNCTION` and then reinstall it with `CREATE FUNCTION`. You would need to do this, for example, if you recompile a new version of your function, so that `mysqld` gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` [392] option. In this case, UDF initialization is skipped and UDFs are unavailable.

18.2.2.6 User-Defined Function Security Precautions

MySQL takes several measures to prevent misuse of user-defined functions.

UDF object files cannot be placed in arbitrary directories. They must be located in some system directory that the dynamic linker is configured to search. To enforce this restriction and prevent attempts at specifying path names outside of directories searched by the dynamic linker, MySQL checks the shared object file name specified in `CREATE FUNCTION` statements for path name delimiter characters. As of MySQL 4.0.24 and 4.1.10a, MySQL also checks for path name delimiters in file names stored in the `mysql.func` table when it loads functions. This prevents attempts at specifying illegitimate path names through direct manipulation of the `mysql.func` table. For information about UDFs and the runtime linker, see Section 18.2.2.5, “Compiling and Installing User-Defined Functions”.

To use `CREATE FUNCTION` or `DROP FUNCTION`, you must have the `INSERT` [492] or `DELETE` [491] privilege, respectively, for the `mysql` database. This is necessary because those statements add and delete rows from the `mysql.func` table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. As of MySQL 4.0.24 and 4.1.10a, `mysqld` supports an `--allow-suspicious-udfs` [384] option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legitimate UDFs. If you have older UDFs that contain only the `xxx`

symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs` [384] option. Otherwise, you should avoid enabling this capability.

18.2.3 Adding a New Native Function

To add a new native MySQL function, use the procedure described here, which requires that you use a source distribution. You cannot add native functions to a binary distribution because it is necessary to modify MySQL source code and compile MySQL from the modified source. If you migrate to another version of MySQL (for example, when a new version is released), you must repeat the procedure with the new version.

If the new native function will be referred to in statements that will be replicated to slave servers, you must ensure that every slave server also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

To add a new native function, follow these steps to modify source files in the `sql` directory:

1. Add one line to `lex.h` that defines the function name in the `sql_functions[]` array.

2. If the function prototype is simple (just takes zero, one, two, or three arguments), add a line to the `sql_functions[]` array in `lex.h` that specifies `SYM(FUNC_ARGN)` as the second argument (where *`N`* is the number of arguments the function takes). Also, add a function in `item_create.cc` that creates a function object. Look at `"ABS"` and `create_funcs_abs()` for an example of this.

   If the function prototype is not simple (for example, if it takes a variable number of arguments), you should make two changes to `sql_yacc.yy`. One is a line that indicates the preprocessor symbol that `yacc` should define; this should be added at the beginning of the file. The other is an “item” to be added to the `simple_expr` parsing rule that defines the function parameters. You will need an item for each syntax with which the function can be called. For an example that shows how this is done, check all occurrences of `ATAN` in `sql_yacc.yy`.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.

4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

   ```
   double   Item_func_newname::val()
   longlong Item_func_newname::val_int()
   String  *Item_func_newname::Str(String *str)
   ```

   If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. If the function is nondeterministic, include the following statement in the item constructor to indicate that function results should not be cached:

   ```
   current_thd->lex->safe_to_cache_query=0;
   ```

   A function is nondeterministic if, given fixed values for its arguments, it can return different results for different invocations.

6. You should probably also define the following object function:

```
void Item_func_newname::fix_length_and_dec()
```

This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. Look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, do not use any global or static variables in the functions without protecting them with mutexes.

If you want to return `NULL` from `::val()`, `::val_int()`, or `::str()`, you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)
- The `::str()` function should return the string that holds the result, or `(char*) 0` if the result is `NULL`.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

18.3 Adding New Procedures to MySQL

In MySQL, you can define a procedure in C++ that can access and modify the data in a query before it is sent to the client. The modification can be done on a row-by-row or `GROUP BY` level.

We have created an example procedure in MySQL 3.23 to show you what can be done.

18.3.1 PROCEDURE ANALYSE

`ANALYSE([max_elements[,max_memory]])`

`ANALYSE()` is defined in the `sql/sql_analyse.cc` source file, which serves as an example of how to create a procedure for use with the `PROCEDURE` clause of `SELECT` statements. `ANALYSE()` is built in and is available by default; other procedures can be created using the format demonstrated in the source file.

`ANALYSE()` examines the result from a query and returns an analysis of the results that suggests optimal data types for each column that may help reduce table sizes. To obtain this analysis, append `PROCEDURE ANALYSE` to the end of a `SELECT` statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that `PROCEDURE ANALYSE()` does not suggest the `ENUM` data type when it is not appropriate.

The arguments are optional and are used as follows:

- *`max_elements`* (default 256) is the maximum number of distinct values that `ANALYSE()` notices per column. This is used by `ANALYSE()` to check whether the optimal data type should be of type `ENUM`; if there are more than *`max_elements`* distinct values, then `ENUM` is not a suggested type.
- *`max_memory`* (default 8192) is the maximum amount of memory that `ANALYSE()` should allocate per column while trying to find all distinct values.

18.3.2 Writing a Procedure

You can find information about procedures by examining the following source files:

- `sql/sql_analyse.cc`
- `sql/procedure.h`
- `sql/procedure.cc`
- `sql/sql_select.cc`

See also MySQL Internals: Writing a Procedure.

18.4 Porting to Other Systems

This appendix helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See Section 2.1.1, “Operating Systems On Which MySQL Is Known To Run”. If you have created a new port of MySQL, please let us know so that we can list it here and on our Web site (http://www.mysql.com/), recommending it to other users.

Note

If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working POSIX thread library is needed for the server. On Solaris 2.5 we use Sun PThreads (the native thread support in 2.4 and earlier versions is not good enough), on Linux we use LinuxThreads by Xavier Leroy, `<Xavier.Leroy@inria.fr>`.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See `mit-pthreads/README` and Programming POSIX Threads (http://www.humanfactor.com/pthreads/).

Up to MySQL 4.0.2, the MySQL distribution included a patched version of Chris Provenzano's Pthreads from MIT (see the MIT Pthreads Web page at http://www.mit.edu/afs/sipb/project/pthreads/ and a programming introduction at http://www.mit.edu:8001/people/proven/IAP_2000/). These can be used for some operating systems that do not have POSIX threads. See Section 2.9.6, “MIT-pthreads Notes”.

It is also possible to use another user level thread package named FSU Pthreads (see http://moss.csc.ncsu.edu/~mueller/pthreads/). This implementation is being used for the SCO port.

See the `thr_lock.c` and `thr_alarm.c` programs in the `mysys` directory for some tests/examples of these problems.

Both the server and the client need a working C++ compiler. We use `gcc` on many platforms. Other compilers that are known to work are Sun Studio, HP-UX `aCC`, IBM AIX `xlC_r`), Intel `ecc/icc`. With previous versions on the respective platforms, we also used Irix `cc` and Compaq `cxx`.

Important

If you are trying to build MySQL 5.1 with `icc` on the IA64 platform, and need support for MySQL Cluster, you should first ensure that you are using `icc` version 9.1.043 or later. (For details, see Bug #21875.)

To compile only the client, use `./configure --without-server`.

If you want or need to change any `Makefile` or the `configure` script, you also need GNU Automake and Autoconf. See Section 2.9.2, “Installing MySQL from a Development Source Tree”.

All steps needed to remake everything from the most basic files.

```
/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug=full --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
gmake clean all install init-db
```

If you run into problems with a new port, you may have to do some debugging of MySQL! See Section 18.4.1, “Debugging a MySQL Server”.

Note

Before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This ensures that your thread installation has even a remote chance to work!

18.4.1 Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` (which disables all new, potentially unsafe functionality) or with `--safe-mode` [391] which disables a lot of optimization that may cause problems. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.

If `mysqld` doesn't want to start, you should verify that you don't have any `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`.

If `mysqld` starts to eat up CPU or memory or if it “hangs,” you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

The command `mysqladmin debug` dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or `myisamchk`. See Chapter 5, *MySQL Server Administration*. You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See Section 2.12, “Operating System-Specific Notes”.

18.4.1.1 Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `--with-debug` [98] or the `--with-debug=full` [98] option. You can check whether MySQL was compiled with debugging by doing: `mysqld --help`. If the `--debug` [385] flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... --debug` in this case.

If you are using `gcc`, the recommended `configure` line is:

```
CC=gcc CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors \
   -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
   --with-debug --with-extra-charsets=complex
```

This avoids problems with the `libstdc++` library and with C++ exceptions (many compilers have problems with C++ exceptions in threaded code) and compile a MySQL version with support for all character sets.

If you suspect a memory overrun error, you can configure MySQL with `--with-debug=full` [98], which installs a memory allocation (`SAFEMALLOC`) checker. However, running with `SAFEMALLOC` is quite slow, so if you get performance problems you should start `mysqld` with the `--skip-safemalloc` [394] option. This disables the memory overrun checks for each call to `malloc()` and `free()`.

If `mysqld` stops crashing when you compile it with `--with-debug` [98], you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` to the `CFLAGS` and `CXXFLAGS` variables above and not use `--with-debug` [98]. If `mysqld` dies, you can at least attach to it with `gdb` or use `gdb` on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something “unexpected,” an entry is written to `stderr`, which `mysqld_safe` directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See Section 1.7.1, “MySQL Mailing Lists”. If you believe that you have found a bug, please use the instructions at Section 1.8, “How to Report Bugs or Problems”.

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

18.4.1.2 Creating Trace Files

If the `mysqld` server doesn't start or if you can cause it to crash quickly, you can try to create a trace file to find the problem.

To do this, you must have a `mysqld` that has been compiled with debugging support. You can check this by executing `mysqld -V`. If the version number ends with `-debug`, it is compiled with support for trace files. (On Windows, the debugging server is named `mysqld-debug` rather than `mysqld` as of MySQL 4.1.)

Start the `mysqld` server with a trace log in `/tmp/mysqld.trace` on Unix or `C:\mysqld.trace` on Windows:

```
```

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` [393] flag to not start `mysqld` as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the `mysqld` server with `mysqladmin shutdown`.

The trace file can become **very large**! To generate a smaller trace file, you can use debugging options something like this:

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace
```

This only prints information with the most interesting tags to the trace file.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can open a bug report and upload the trace file to the report, so that a MySQL developer can take a look at it. For instructions, see Section 1.8, “How to Report Bugs or Problems”.

The trace file is made with the **DBUG** package by Fred Fish. See Section 18.4.3, “The DBUG Package”.

18.4.1.3 Debugging `mysqld` under `gdb`

On most systems you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some older `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case, you can only have one thread active at a time. It is best to upgrade to `gdb` 5.1 because thread debugging works much better with this version!

NPTL threads (the new thread library on Linux) may cause problems while running `mysqld` under `gdb`. Some symptoms are:

- `mysqld` hangs during startup (before it writes `ready for connections`).
- `mysqld` crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case, you should set the following environment variable in the shell before starting `gdb`:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running `mysqld` under `gdb`, you should disable the stack trace with `--skip-stack-trace` [394] to be able to catch segfaults within `gdb`.

In MySQL 4.0.14 and above you should use the `--gdb` [387] option to `mysqld`. This installs an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling.

It is very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` doesn't free the memory for old threads. You can avoid this problem by starting `mysqld` with `thread_cache_size` [431] set to a value equal to `max_connections` [419] + 1. In most cases just using `--thread_cache_size=5'` [431] helps a lot!

If you want to get a core dump on Linux if `mysqld` dies with a SIGSEGV signal, you can start `mysqld` with the `--core-file` [385] option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   quit
```

See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.

If you are using `gdb` 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with `gdb`, you should download gdb 5.x and try this instead. The new `gdb` version has very improved thread handling!

Here is an example how to debug mysqld:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the above output in a bug report, which you can file using the instructions in Section 1.8, “How to Report Bugs or Problems”.

If `mysqld` hangs you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

18.4.1.4 Using a Stack Trace

On some operating systems, the error log contains a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See Section 5.3.1, “The Error Log”. To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to gcc. See Section 18.4.1.1, “Compiling MySQL for Debugging”.

A stack trace in the error log looks something like this:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
```

```
this, something went terribly wrong...

stack range sanity check, ok, backtrace follows
0x40077552
0x81281a0
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

You can use the `resolve_stack_dump` utility to determine where `mysqld` died by using the following procedure:

1. Copy the preceding numbers to a file, for example `mysqld.stack`:

   ```
   0x9da402
   0x6648e9
   0x7f1a5af000f0
   0x7f1a5a10f0f2
   0x7412cb
   0x688354
   0x688494
   0x67a170
   0x67f0ad
   0x67fdf8
   0x6811b6
   0x66e05e
   ```

2. Make a symbol file for the `mysqld` server:

   ```
   shell> nm -n libexec/mysqld > /tmp/mysqld.sym
   ```

   If `mysqld` is not linked statically, use the following command instead:

   ```
   shell> nm -D -n libexec/mysqld > /tmp/mysqld.sym
   ```

   If you want to decode C++ symbols, use the `--demangle`, if available, to `nm`. If your version of `nm` does not have this option, you will need to use the `c++filt` command after the stack dump has been produced to demangle the C++ names.

   Note that most MySQL binary distributions (except for the "debug" packages, where this information is included inside of the binaries themselves) ship with the above file, named `mysqld.sym.gz`. In this case, you can simply unpack it like this:

   ```
   shell> gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
   ```

3. Execute the following command:

   ```
   shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack
   ```

   If you were not able to include demangled C++ names in your symbol file, process the `resolve_stack_dump` output using `c++filt`:

```
shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack | c++filt
```

This prints out where `mysqld` died. If that does not help you find out why `mysqld` died, you should create a bug report and include the output from the preceding command with the bug report.

However, in most cases it does not help us to have just a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, in most cases we need to know the statement that killed `mysqld` and preferably a test case so that we can repeat the problem! See Section 1.8, "How to Report Bugs or Problems".

18.4.1.5 Using Server Logs to Find Causes of Errors in `mysqld`

Note that before starting `mysqld` with `--log` [388] you should check all your tables with `myisamchk`. See Chapter 5, *MySQL Server Administration*.

If `mysqld` dies or hangs, you should start `mysqld` with `--log` [388]. When `mysqld` dies again, you can examine the end of the log file for the query that killed `mysqld`.

If you are using `--log` [388] without a file name, the log is stored in the database directory as *`host_name`*`.log` In most cases it is the last query in the log file that killed `mysqld`, but if possible you should verify this by restarting `mysqld` and executing the found query from the `mysql` command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that `mysqld` is using indexes properly. See Section 12.7.2, "`EXPLAIN` Syntax".

You can find the queries that take a long time to execute by starting `mysqld` with `--log-slow-queries` [388]. See Section 5.3.5, "The Slow Query Log".

If you find the text `mysqld restarted` in the error log file (normally named `hostname.err`) you probably have found a query that causes `mysqld` to fail. If this happens, you should check all your tables with `myisamchk` (see Chapter 5, *MySQL Server Administration*), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to a MySQL mailing list. The mailing lists are described at http://lists.mysql.com/, which also has links to online list archives.

If you have started `mysqld` with `--myisam-recover` [390], MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further. See Section 5.1.2, "Server Command Options".

It is not a good sign if `mysqld` did die unexpectedly, but in this case, you should not investigate the `Checking table...` messages, but instead try to find out why `mysqld` died.

18.4.1.6 Making a Test Case If You Experience Table Corruption

If you get corrupted tables or if `mysqld` always fails after some update commands, you can test whether this bug is reproducible by doing the following:

- Take down the MySQL daemon (with `mysqladmin shutdown`).
- Make a backup of the tables (to guard against the very unlikely case that the repair does something bad).

- Check all tables with `myisamchk -s database/*.MYI`. Repair any wrong tables with `myisamchk -r database/table.MYI`.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start `mysqld` with `--log-bin` [1181]. See Section 5.3.4, "The Binary Log". If you want to find a query that crashes `mysqld`, you should use `--log` [388] `--log-bin` [1181].
- When you have gotten a crashed table, stop the `mysqld server`.
- Restore the backup.
- Restart the `mysqld` server **without** `--log-bin` [1181]
- Re-execute the commands with `mysqlbinlog binary-log-file | mysql`. The binary log is saved in the MySQL database directory with the name `hostname-bin.#`.
- If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found reproducible bug that should be easy to fix! FTP the tables and the binary log to our bugs database using the instructions given in Section 1.8, "How to Report Bugs or Problems". If you are a support customer, you can use the MySQL Customer Support Center https://support.mysql.com/ to alert the MySQL team about the problem and have it fixed as soon as possible.

You can also use the script `mysql_find_rows` to just execute some of the update statements if you want to narrow down the problem.

18.4.2 Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `--with-debug` [98] or `--with-debug=full` [98]. See Section 2.9.3, "MySQL Source-Configuration Options".

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See Section 1.8, "How to Report Bugs or Problems".

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

18.4.3 The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is debugging. See Section 18.4.1.2, “Creating Trace Files”.

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support. For more information about programming with the DBUG package, see the DBUG manual in the `dbug` directory of MySQL source distributions. It is best to use a recent distribution to get the most updated DBUG manual.

You use the debug package by invoking a program with the `--debug="..."` or the `-#...` option.

Most MySQL programs have a default debug string that is used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/program_name.trace` on Unix and `\program_name.trace` on Windows.

The debug control string is a sequence of colon-separated fields as follows:

```
<field_1>:<field_2>:...:<field_N>
```

Each field consists of a mandatory flag character followed by an optional “`,`” and comma-separated list of modifiers:

```
flag[,modifier,modifier,...,modifier]
```

The following table shows the currently recognized flag characters.

Flag	Description
`d`	Enable output from DBUG_<N> macros for the current state. May be followed by a list of keywords which selects output only for the DBUG macros with that keyword. An empty list of keywords implies output for all macros.
`D`	Delay after each debugger output line. The argument is the number of tenths of seconds to delay, subject to machine capabilities. For example, `-#D,20` specifies a delay of two seconds.
`f`	Limit debugging, tracing, and profiling to the list of named functions. Note that a null list disables all functions. The appropriate `d` or `t` flags must still be given; this flag only limits their actions if they are enabled.
`F`	Identify the source file name for each line of debug or trace output.
`i`	Identify the process with the PID or thread ID for each line of debug or trace output.
`g`	Enable profiling. Create a file called `dbugmon.out` containing information that can be used to profile the program. May be followed by a list of keywords that select profiling only for the functions in that list. A null list implies that all functions are considered.
`L`	Identify the source file line number for each line of debug or trace output.
`n`	Print the current function nesting depth for each line of debug or trace output.
`N`	Number each line of debug output.
`o`	Redirect the debugger output stream to the specified file. The default output is `stderr`.
`O`	Like `o`, but the file is really flushed between each write. When needed, the file is closed and reopened between each write.
`p`	Limit debugger actions to specified processes. A process must be identified with the `DBUG_PROCESS` macro and match one in the list for debugger actions to occur.
`P`	Print the current process name for each line of debug or trace output.

Flag	Description
r	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
S	Do function _sanity(_file_,_line_) at each debugged function until _sanity() returns something that differs from 0. (Mostly used with safemalloc to find memory leaks)
t	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

Some examples of debug control strings that might appear on a shell command line (the -# is typically used to introduce a control string to an application program) are:

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:O,\\mysqld.trace
```

In MySQL, common tags to print (with the d option) are enter, exit, error, warning, info, and loop.

Appendix A Licenses for Third-Party Components

Table of Contents

The following is a list of the creators of the libraries we have included with the MySQL server source to make it easy to compile and install MySQL. We are thankful to all individuals that have created these. Some of these libraries require that their licensing terms be included in the documentation of products that include them. Cross references to these licensing terms are given with the applicable items in the list.

- Fred Fish

  For his excellent C debugging and trace library. Monty has made a number of smaller improvements to the library (speed and additional options).

- Richard A. O'Keefe

  For his public domain string library.

- Jean-loup Gailly and Mark Adler

  For the `zlib` library, used on MySQL on Windows and on platforms where the host `zlib` is too old.

- Bjorn Benson

  For his safe_malloc (memory checker) package which is used in when you build MySQL using one of the `BUILD/compile-*-debug` scripts or by manually setting the `-DSAFEMALLOC` flag.

- Free Software Foundation

  The `readline` library, used by the `mysql` command-line client.

- The NetBSD Foundation

  The `libedit` library, optionally used by the `mysql` command-line client. `libedit` is used for commercial builds because `readline` is covered under the GPL. License: Section A.3, “Editline Library (`libedit`) License”

A.1 RegEX-Spencer Library License

The following software may be included in this product: Henry Spencer's Regular-Expression Library (RegEX-Spencer)

```
Copyright 1992, 1993, 1994 Henry Spencer.  All rights reserved.
This software is not subject to any license of the American Telephone
and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on
any computer system, and to alter it and redistribute it, subject
to the following restrictions:

1. The author is not responsible for the consequences of use of this
```

```
   software, no matter how awful, even if they arise from flaws in it.

2. The origin of this software must not be misrepresented, either by
   explicit claim or by omission.  Since few users ever read sources,
   credits must appear in the documentation.

3. Altered versions must be plainly marked as such, and must not be
   misrepresented as being the original software.  Since few users
   ever read sources, credits must appear in the documentation.

4. This notice may not be removed or altered.
```

A.2 RSA MD5 Algorithm License

The RSA Data Security, Inc. MD5 Message-Digest Algorithm (“MD5 algorithm”) is covered by this license:

```
/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
*/
```

A.3 Editline Library (libedit) License

The following software may be included in this product:

Editline Library (libedit)

Some files are:

```
Copyright (c) 1992, 1993
The Regents of the University of California. All rights reserved.

This code is derived from software contributed to
Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the
  above copyright notice, this list of conditions
  and the following disclaimer.
2. Redistributions in binary form must reproduce the
  above copyright notice, this list of conditions and
  the following disclaimer in the documentation and/or
```

```
  other materials provided with the distribution.
3. Neither the name of the University nor the names of
  its contributors may be used to endorse or promote
  products derived from this software without specific
  prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

Some files are:

```

Copyright (c) 2001 The NetBSD Foundation, Inc.
All rights reserved.

This code is derived from software contributed to The NetBSD Foundation

by Anthony Mallet.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the
   above copyright notice, this list of conditions
   and the following disclaimer.
2. Redistributions in binary form must reproduce the
    above copyright notice, this list of conditions and the
    following disclaimer in the documentation and/or
    other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC.
AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED  TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

Some files are:

```

Copyright (c) 1997 The NetBSD Foundation, Inc.
All rights reserved.

This code is derived from software contributed to The NetBSD Foundation

by Jaromir Dolecek.
```

```
Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the
   above copyright notice, this list of conditions
   and the following disclaimer.
2. Redistributions in binary form must reproduce
    the above copyright notice, this list of conditions
    and the following disclaimer in the documentation
    and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC.
AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED  TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

Some files are:

```
Copyright (c) 1998 Todd C. Miller <Todd.Miller@courtesan.com>

Permission to use, copy, modify, and distribute this
software for any purpose with or without fee is hereby
granted, provided that the above copyright notice and
this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND TODD C. MILLER
DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL TODD C. MILLER BE LIABLE
FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

Appendix B Errors, Error Codes, and Common Problems

Table of Contents

This appendix lists common problems and errors that may occur and potential resolutions, in addition to listing the errors that may appear when you call MySQL from any host language. The first section covers problems and resolutions. Detailed information on errors is provided: One list displays server error messages. Another list displays client program messages.

B.1 Sources of Error Information

There are several sources of error information in MySQL:

- Each SQL statement executed results in an error code, an SQLSTATE value, and an error message, as described in Section B.2, “Types of Error Values”. These errors are returned from the server side; see Section B.3, “Server Error Codes and Messages”.
- Errors can occur on the client side, usually involving problems communicating with the server; see Section B.4, “Client Error Codes and Messages”.
- SQL statement warning and error information is available through the `SHOW WARNINGS` and `SHOW ERRORS` statements. The `warning_count` [434] system variable indicates the number of errors, warnings, and notes. The `error_count` [410] system variable indicates the number of errors. Its value excludes warnings and notes.
- `SHOW SLAVE STATUS` statement output includes information about replication errors occurring on the slave side.
- `SHOW ENGINE INNODB STATUS` statement output includes information about the most recent foreign key error if a `CREATE TABLE` statement for an `InnoDB` table fails.
- The `perror` program provides information from the command line about error numbers. See Section 4.8.1, “`perror` — Explain Error Codes”.

Descriptions of server and client errors are provided later in this Appendix. For information about errors related to `InnoDB`, see Section 13.2.13, “`InnoDB` Error Handling”.

B.2 Types of Error Values

When an error occurs in MySQL, the server returns two types of error values:

- A MySQL-specific error code. This value is numeric. It is not portable to other database systems.
- An SQLSTATE value. The value is a five-character string (for example, `'42S02'`). The values are taken from ANSI SQL and ODBC and are more standardized.

A message string that provides a textual description of the error is also available.

When an error occurs, the MySQL error code, SQLSTATE value, and message string are available using C API functions:

- MySQL error code: Call `mysql_errno()`
- SQLSTATE value: Call `mysql_sqlstate()`
- Error message: Call `mysql_error()`

For prepared statements, the corresponding error functions are `mysql_stmt_errno()`, `mysql_stmt_sqlstate()`, and `mysql_stmt_error()`. All error functions are described in Section 17.6, “MySQL C API”.

The number of errors, warnings, and notes for the previous statement can be obtained by calling `mysql_warning_count()`. See Section 17.6.6.70, “`mysql_warning_count()`”.

The first two characters of an SQLSTATE value indicate the error class:

- Class = `'00'` indicates success.
- Class = `'01'` indicates a warning.
- Class = `'02'` indicates “not found.” This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
- Class > `'02'` indicates an exception.

B.3 Server Error Codes and Messages

MySQL programs have access to several types of error information when the server returns an error. For example, the `mysql` client program displays errors using the following format:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

The message displayed contains three types of information:

- A numeric error code (`1146`). This number is MySQL-specific and is not portable to other database systems.
- A five-character SQLSTATE value (`'42S02'`). The values are taken from ANSI SQL and ODBC and are more standardized. Not all MySQL error numbers have corresponding SQLSTATE values. In these cases, `'HY000'` (general error) is used.
- A message string that provides a textual description of the error.

Server error information comes from the following source files. For details about the way that error information is defined, see the MySQL Internals manual, available at http://dev.mysql.com/doc/.

- The Error values and the symbols in parentheses correspond to definitions in the `include/mysqld_error.h` MySQL source file.

- The SQLSTATE values correspond to definitions in the `include/sql_state.h` MySQL source file.

  SQLSTATE error codes are displayed only if you use MySQL version 4.1 and up. SQLSTATE codes were added for compatibility with X/Open, ANSI, and ODBC behavior.

- The Message values correspond to the error messages that are listed in the `sql/share/errmsg.txt` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: `1000` SQLSTATE: `HY000` (`ER_HASHCHK` [1567])

  Message: hashchk

- Error: `1001` SQLSTATE: `HY000` (`ER_NISAMCHK` [1567])

  Message: isamchk

- Error: `1002` SQLSTATE: `HY000` (`ER_NO` [1567])

  Message: NO

- Error: `1003` SQLSTATE: `HY000` (`ER_YES` [1567])

  Message: YES

- Error: `1004` SQLSTATE: `HY000` (`ER_CANT_CREATE_FILE` [1567])

  Message: Can't create file '%s' (errno: %d)

- Error: `1005` SQLSTATE: `HY000` (`ER_CANT_CREATE_TABLE` [1567])

  Message: Can't create table '%s' (errno: %d)

- Error: `1006` SQLSTATE: `HY000` (`ER_CANT_CREATE_DB` [1567])

  Message: Can't create database '%s' (errno: %d)

- Error: `1007` SQLSTATE: `HY000` (`ER_DB_CREATE_EXISTS` [1567])

  Message: Can't create database '%s'; database exists

- Error: `1008` SQLSTATE: `HY000` (`ER_DB_DROP_EXISTS` [1567])

  Message: Can't drop database '%s'; database doesn't exist

- Error: `1009` SQLSTATE: `HY000` (`ER_DB_DROP_DELETE` [1567])

  Message: Error dropping database (can't delete '%s', errno: %d)

- Error: `1010` SQLSTATE: `HY000` (`ER_DB_DROP_RMDIR` [1567])

  Message: Error dropping database (can't rmdir '%s', errno: %d)

- Error: `1011` SQLSTATE: `HY000` (`ER_CANT_DELETE_FILE` [1567])

  Message: Error on delete of '%s' (errno: %d)

- Error: 1012 SQLSTATE: HY000 (ER_CANT_FIND_SYSTEM_REC [1568])

  Message: Can't read record in system table

- Error: 1013 SQLSTATE: HY000 (ER_CANT_GET_STAT [1568])

  Message: Can't get status of '%s' (errno: %d)

- Error: 1014 SQLSTATE: HY000 (ER_CANT_GET_WD [1568])

  Message: Can't get working directory (errno: %d)

- Error: 1015 SQLSTATE: HY000 (ER_CANT_LOCK [1568])

  Message: Can't lock file (errno: %d)

- Error: 1016 SQLSTATE: HY000 (ER_CANT_OPEN_FILE [1568])

  Message: Can't open file: '%s' (errno: %d)

- Error: 1017 SQLSTATE: HY000 (ER_FILE_NOT_FOUND [1568])

  Message: Can't find file: '%s' (errno: %d)

- Error: 1018 SQLSTATE: HY000 (ER_CANT_READ_DIR [1568])

  Message: Can't read dir of '%s' (errno: %d)

- Error: 1019 SQLSTATE: HY000 (ER_CANT_SET_WD [1568])

  Message: Can't change dir to '%s' (errno: %d)

- Error: 1020 SQLSTATE: HY000 (ER_CHECKREAD [1568])

  Message: Record has changed since last read in table '%s'

- Error: 1021 SQLSTATE: HY000 (ER_DISK_FULL [1568])

  Message: Disk full (%s); waiting for someone to free some space...

- Error: 1022 SQLSTATE: 23000 (ER_DUP_KEY [1568])

  Message: Can't write; duplicate key in table '%s'

- Error: 1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE [1568])

  Message: Error on close of '%s' (errno: %d)

- Error: 1024 SQLSTATE: HY000 (ER_ERROR_ON_READ [1568])

  Message: Error reading file '%s' (errno: %d)

- Error: 1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME [1568])

  Message: Error on rename of '%s' to '%s' (errno: %d)

- Error: 1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE [1568])

  Message: Error writing file '%s' (errno: %d)

- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED [1569])

  Message: '%s' is locked against change

- Error: 1028 SQLSTATE: HY000 (ER_FILSORT_ABORT [1569])

  Message: Sort aborted

- Error: 1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND [1569])

  Message: View '%s' doesn't exist for '%s'

- Error: 1030 SQLSTATE: HY000 (ER_GET_ERRNO [1569])

  Message: Got error %d from storage engine

- Error: 1031 SQLSTATE: HY000 (ER_ILLEGAL_HA [1569])

  Message: Table storage engine for '%s' doesn't have this option

- Error: 1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND [1569])

  Message: Can't find record in '%s'

- Error: 1033 SQLSTATE: HY000 (ER_NOT_FORM_FILE [1569])

  Message: Incorrect information in file: '%s'

- Error: 1034 SQLSTATE: HY000 (ER_NOT_KEYFILE [1569])

  Message: Incorrect key file for table '%s'; try to repair it

- Error: 1035 SQLSTATE: HY000 (ER_OLD_KEYFILE [1569])

  Message: Old key file for table '%s'; repair it!

- Error: 1036 SQLSTATE: HY000 (ER_OPEN_AS_READONLY [1569])

  Message: Table '%s' is read only

- Error: 1037 SQLSTATE: HY001 (ER_OUTOFMEMORY [1569])

  Message: Out of memory; restart server and try again (needed %d bytes)

- Error: 1038 SQLSTATE: HY001 (ER_OUT_OF_SORTMEMORY [1569])

  Message: Out of sort memory; increase server sort buffer size

- Error: 1039 SQLSTATE: HY000 (ER_UNEXPECTED_EOF [1569])

  Message: Unexpected EOF found when reading file '%s' (errno: %d)

- Error: 1040 SQLSTATE: 08004 (ER_CON_COUNT_ERROR [1569])

  Message: Too many connections

- Error: 1041 SQLSTATE: HY000 (ER_OUT_OF_RESOURCES [1569])

  Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

- Error: 1042 SQLSTATE: 08S01 (ER_BAD_HOST_ERROR [1570])

  Message: Can't get hostname for your address

- Error: 1043 SQLSTATE: 08S01 (ER_HANDSHAKE_ERROR [1570])

  Message: Bad handshake

- Error: 1044 SQLSTATE: 42000 (ER_DBACCESS_DENIED_ERROR [1570])

  Message: Access denied for user '%s'@'%s' to database '%s'

- Error: 1045 SQLSTATE: 28000 (ER_ACCESS_DENIED_ERROR [1570])

  Message: Access denied for user '%s'@'%s' (using password: %s)

- Error: 1046 SQLSTATE: 3D000 (ER_NO_DB_ERROR [1570])

  Message: No database selected

- Error: 1047 SQLSTATE: 08S01 (ER_UNKNOWN_COM_ERROR [1570])

  Message: Unknown command

- Error: 1048 SQLSTATE: 23000 (ER_BAD_NULL_ERROR [1570])

  Message: Column '%s' cannot be null

- Error: 1049 SQLSTATE: 42000 (ER_BAD_DB_ERROR [1570])

  Message: Unknown database '%s'

- Error: 1050 SQLSTATE: 42S01 (ER_TABLE_EXISTS_ERROR [1570])

  Message: Table '%s' already exists

- Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR [1570])

  Message: Unknown table '%s'

- Error: 1052 SQLSTATE: 23000 (ER_NON_UNIQ_ERROR [1570])

  Message: Column '%s' in %s is ambiguous

- Error: 1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN [1570])

  Message: Server shutdown in progress

- Error: 1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR [1570])

  Message: Unknown column '%s' in '%s'

- Error: 1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP [1570])

  Message: '%s' isn't in GROUP BY

- Error: 1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD [1570])

  Message: Can't group on '%s'

- Error: 1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT [1571])

  Message: Statement has sum functions and columns in same statement
- Error: 1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT [1571])

  Message: Column count doesn't match value count
- Error: 1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT [1571])

  Message: Identifier name '%s' is too long
- Error: 1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME [1571])

  Message: Duplicate column name '%s'
- Error: 1061 SQLSTATE: 42000 (ER_DUP_KEYNAME [1571])

  Message: Duplicate key name '%s'
- Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY [1571])

  Message: Duplicate entry '%s' for key %d
- Error: 1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC [1571])

  Message: Incorrect column specifier for column '%s'
- Error: 1064 SQLSTATE: 42000 (ER_PARSE_ERROR [1571])

  Message: %s near '%s' at line %d
- Error: 1065 SQLSTATE: HY000 (ER_EMPTY_QUERY [1571])

  Message: Query was empty
- Error: 1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE [1571])

  Message: Not unique table/alias: '%s'
- Error: 1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT [1571])

  Message: Invalid default value for '%s'
- Error: 1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY [1571])

  Message: Multiple primary key defined
- Error: 1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS [1571])

  Message: Too many keys specified; max %d keys allowed
- Error: 1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS [1571])

  Message: Too many key parts specified; max %d parts allowed
- Error: 1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY [1571])

  Message: Specified key was too long; max key length is %d bytes

- Error: 1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXITS [1572])

  Message: Key column '%s' doesn't exist in table

- Error: 1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY [1572])

  Message: BLOB column '%s' can't be used in key specification with the used table type

- Error: 1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH [1572])

  Message: Column length too big for column '%s' (max = %d); use BLOB or TEXT instead

- Error: 1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY [1572])

  Message: Incorrect table definition; there can be only one auto column and it must be defined as a key

- Error: 1076 SQLSTATE: HY000 (ER_READY [1572])

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d

- Error: 1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN [1572])

  Message: %s: Normal shutdown

- Error: 1078 SQLSTATE: HY000 (ER_GOT_SIGNAL [1572])

  Message: %s: Got signal %d. Aborting!

- Error: 1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE [1572])

  Message: %s: Shutdown complete

- Error: 1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE [1572])

  Message: %s: Forcing close of thread %ld user: '%s'

- Error: 1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR [1572])

  Message: Can't create IP socket

- Error: 1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX [1572])

  Message: Table '%s' has no index like the one used in CREATE INDEX; recreate the table

- Error: 1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS [1572])

  Message: Field separator argument is not what is expected; check the manual

- Error: 1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED [1572])

  Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'

- Error: 1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE [1572])

  Message: The file '%s' must be in the database directory or be readable by all

- Error: 1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR [1572])

  Message: File '%s' already exists

- Error: 1087 SQLSTATE: HY000 (ER_LOAD_INFO [1573])

  Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld

- Error: 1088 SQLSTATE: HY000 (ER_ALTER_INFO [1573])

  Message: Records: %ld Duplicates: %ld

- Error: 1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY [1573])

  Message: Incorrect sub part key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique sub keys

- Error: 1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS [1573])

  Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead

- Error: 1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY [1573])

  Message: Can't DROP '%s'; check that column/key exists

- Error: 1092 SQLSTATE: HY000 (ER_INSERT_INFO [1573])

  Message: Records: %ld Duplicates: %ld Warnings: %ld

- Error: 1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED [1573])

  Message: You can't specify target table '%s' for update in FROM clause

- Error: 1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD [1573])

  Message: Unknown thread id: %lu

- Error: 1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR [1573])

  Message: You are not owner of thread %lu

- Error: 1096 SQLSTATE: HY000 (ER_NO_TABLES_USED [1573])

  Message: No tables used

- Error: 1097 SQLSTATE: HY000 (ER_TOO_BIG_SET [1573])

  Message: Too many strings for column %s and SET

- Error: 1098 SQLSTATE: HY000 (ER_NO_UNIQUE_LOGFILE [1573])

  Message: Can't generate a unique log-filename %s.(1-999)

- Error: 1099 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED_FOR_WRITE [1573])

  Message: Table '%s' was locked with a READ lock and can't be updated

- Error: 1100 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED [1573])

  Message: Table '%s' was not locked with LOCK TABLES

- Error: 1101 SQLSTATE: 42000 (ER_BLOB_CANT_HAVE_DEFAULT [1573])

  Message: BLOB/TEXT column '%s' can't have a default value

- Error: 1102 SQLSTATE: 42000 (ER_WRONG_DB_NAME [1574])

  Message: Incorrect database name '%s'

- Error: 1103 SQLSTATE: 42000 (ER_WRONG_TABLE_NAME [1574])

  Message: Incorrect table name '%s'

- Error: 1104 SQLSTATE: 42000 (ER_TOO_BIG_SELECT [1574])

  Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is okay

- Error: 1105 SQLSTATE: HY000 (ER_UNKNOWN_ERROR [1574])

  Message: Unknown error

- Error: 1106 SQLSTATE: 42000 (ER_UNKNOWN_PROCEDURE [1574])

  Message: Unknown procedure '%s'

- Error: 1107 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_PROCEDURE [1574])

  Message: Incorrect parameter count to procedure '%s'

- Error: 1108 SQLSTATE: HY000 (ER_WRONG_PARAMETERS_TO_PROCEDURE [1574])

  Message: Incorrect parameters to procedure '%s'

- Error: 1109 SQLSTATE: 42S02 (ER_UNKNOWN_TABLE [1574])

  Message: Unknown table '%s' in %s

- Error: 1110 SQLSTATE: 42000 (ER_FIELD_SPECIFIED_TWICE [1574])

  Message: Column '%s' specified twice

- Error: 1111 SQLSTATE: HY000 (ER_INVALID_GROUP_FUNC_USE [1574])

  Message: Invalid use of group function

- Error: 1112 SQLSTATE: 42000 (ER_UNSUPPORTED_EXTENSION [1574])

  Message: Table '%s' uses an extension that doesn't exist in this MySQL version

- Error: 1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS [1574])

  Message: A table must have at least 1 column

- Error: 1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL [1574])

  Message: The table '%s' is full

- Error: 1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET [1574])

  Message: Unknown character set: '%s'

- Error: 1116 SQLSTATE: HY000 (ER_TOO_MANY_TABLES [1574])

  Message: Too many tables; MySQL can only use %d tables in a join

- Error: 1117 SQLSTATE: HY000 (ER_TOO_MANY_FIELDS [1575])

  Message: Too many columns

- Error: 1118 SQLSTATE: 42000 (ER_TOO_BIG_ROWSIZE [1575])

  Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs

- Error: 1119 SQLSTATE: HY000 (ER_STACK_OVERRUN [1575])

  Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld -O thread_stack=#' to specify a bigger stack if needed

- Error: 1120 SQLSTATE: 42000 (ER_WRONG_OUTER_JOIN [1575])

  Message: Cross dependency found in OUTER JOIN; examine your ON conditions

- Error: 1121 SQLSTATE: 42000 (ER_NULL_COLUMN_IN_INDEX [1575])

  Message: Column '%s' is used with UNIQUE or INDEX but is not defined as NOT NULL

- Error: 1122 SQLSTATE: HY000 (ER_CANT_FIND_UDF [1575])

  Message: Can't load function '%s'

- Error: 1123 SQLSTATE: HY000 (ER_CANT_INITIALIZE_UDF [1575])

  Message: Can't initialize function '%s'; %s

- Error: 1124 SQLSTATE: HY000 (ER_UDF_NO_PATHS [1575])

  Message: No paths allowed for shared library

- Error: 1125 SQLSTATE: HY000 (ER_UDF_EXISTS [1575])

  Message: Function '%s' already exists

- Error: 1126 SQLSTATE: HY000 (ER_CANT_OPEN_LIBRARY [1575])

  Message: Can't open shared library '%s' (errno: %d %s)

- Error: 1127 SQLSTATE: HY000 (ER_CANT_FIND_DL_ENTRY [1575])

  Message: Can't find function '%s' in library

- Error: 1128 SQLSTATE: HY000 (ER_FUNCTION_NOT_DEFINED [1575])

  Message: Function '%s' is not defined

- Error: 1129 SQLSTATE: HY000 (ER_HOST_IS_BLOCKED [1575])

  Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

- Error: 1130 SQLSTATE: HY000 (ER_HOST_NOT_PRIVILEGED [1575])

  Message: Host '%s' is not allowed to connect to this MySQL server

- Error: 1131 SQLSTATE: 42000 (ER_PASSWORD_ANONYMOUS_USER [1575])

Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords

- Error: 1132 SQLSTATE: 42000 (ER_PASSWORD_NOT_ALLOWED [1576])

  Message: You must have privileges to update tables in the mysql database to be able to change passwords for others

- Error: 1133 SQLSTATE: 42000 (ER_PASSWORD_NO_MATCH [1576])

  Message: Can't find any matching row in the user table

- Error: 1134 SQLSTATE: HY000 (ER_UPDATE_INFO [1576])

  Message: Rows matched: %ld Changed: %ld Warnings: %ld

- Error: 1135 SQLSTATE: HY000 (ER_CANT_CREATE_THREAD [1576])

  Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug

- Error: 1136 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT_ON_ROW [1576])

  Message: Column count doesn't match value count at row %ld

- Error: 1137 SQLSTATE: HY000 (ER_CANT_REOPEN_TABLE [1576])

  Message: Can't reopen table: '%s'

- Error: 1138 SQLSTATE: 42000 (ER_INVALID_USE_OF_NULL [1576])

  Message: Invalid use of NULL value

- Error: 1139 SQLSTATE: 42000 (ER_REGEXP_ERROR [1576])

  Message: Got error '%s' from regexp

- Error: 1140 SQLSTATE: 42000 (ER_MIX_OF_GROUP_FUNC_AND_FIELDS [1576])

  Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error: 1141 SQLSTATE: 42000 (ER_NONEXISTING_GRANT [1576])

  Message: There is no such grant defined for user '%s' on host '%s'

- Error: 1142 SQLSTATE: 42000 (ER_TABLEACCESS_DENIED_ERROR [1576])

  Message: %s command denied to user '%s'@'%s' for table '%s'

- Error: 1143 SQLSTATE: 42000 (ER_COLUMNACCESS_DENIED_ERROR [1576])

  Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error: 1144 SQLSTATE: 42000 (ER_ILLEGAL_GRANT_FOR_TABLE [1576])

  Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used

- Error: 1145 SQLSTATE: 42000 (ER_GRANT_WRONG_HOST_OR_USER [1577])

  Message: The host or user argument to GRANT is too long

- Error: 1146 SQLSTATE: 42S02 (ER_NO_SUCH_TABLE [1577])

  Message: Table '%s.%s' doesn't exist

- Error: 1147 SQLSTATE: 42000 (ER_NONEXISTING_TABLE_GRANT [1577])

  Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: 1148 SQLSTATE: 42000 (ER_NOT_ALLOWED_COMMAND [1577])

  Message: The used command is not allowed with this MySQL version

- Error: 1149 SQLSTATE: 42000 (ER_SYNTAX_ERROR [1577])

  Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use

- Error: 1150 SQLSTATE: HY000 (ER_DELAYED_CANT_CHANGE_LOCK [1577])

  Message: Delayed insert thread couldn't get requested lock for table %s

- Error: 1151 SQLSTATE: HY000 (ER_TOO_MANY_DELAYED_THREADS [1577])

  Message: Too many delayed threads in use

- Error: 1152 SQLSTATE: 08S01 (ER_ABORTING_CONNECTION [1577])

  Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: 1153 SQLSTATE: 08S01 (ER_NET_PACKET_TOO_LARGE [1577])

  Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: 1154 SQLSTATE: 08S01 (ER_NET_READ_ERROR_FROM_PIPE [1577])

  Message: Got a read error from the connection pipe

- Error: 1155 SQLSTATE: 08S01 (ER_NET_FCNTL_ERROR [1577])

  Message: Got an error from fcntl()

- Error: 1156 SQLSTATE: 08S01 (ER_NET_PACKETS_OUT_OF_ORDER [1577])

  Message: Got packets out of order

- Error: 1157 SQLSTATE: 08S01 (ER_NET_UNCOMPRESS_ERROR [1577])

  Message: Couldn't uncompress communication packet

- Error: 1158 SQLSTATE: 08S01 (ER_NET_READ_ERROR [1577])

  Message: Got an error reading communication packets

- Error: 1159 SQLSTATE: 08S01 (ER_NET_READ_INTERRUPTED [1577])

  Message: Got timeout reading communication packets

- Error: 1160 SQLSTATE: 08S01 (ER_NET_ERROR_ON_WRITE [1578])

  Message: Got an error writing communication packets

- Error: 1161 SQLSTATE: 08S01 (ER_NET_WRITE_INTERRUPTED [1578])

  Message: Got timeout writing communication packets

- Error: 1162 SQLSTATE: 42000 (ER_TOO_LONG_STRING [1578])

  Message: Result string is longer than 'max_allowed_packet' bytes

- Error: 1163 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_BLOB [1578])

  Message: The used table type doesn't support BLOB/TEXT columns

- Error: 1164 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_AUTO_INCREMENT [1578])

  Message: The used table type doesn't support AUTO_INCREMENT columns

- Error: 1165 SQLSTATE: HY000 (ER_DELAYED_INSERT_TABLE_LOCKED [1578])

  Message: INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES

- Error: 1166 SQLSTATE: 42000 (ER_WRONG_COLUMN_NAME [1578])

  Message: Incorrect column name '%s'

- Error: 1167 SQLSTATE: 42000 (ER_WRONG_KEY_COLUMN [1578])

  Message: The used storage engine can't index column '%s'

- Error: 1168 SQLSTATE: HY000 (ER_WRONG_MRG_TABLE [1578])

  Message: Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist

- Error: 1169 SQLSTATE: 23000 (ER_DUP_UNIQUE [1578])

  Message: Can't write, because of unique constraint, to table '%s'

- Error: 1170 SQLSTATE: 42000 (ER_BLOB_KEY_WITHOUT_LENGTH [1578])

  Message: BLOB/TEXT column '%s' used in key specification without a key length

- Error: 1171 SQLSTATE: 42000 (ER_PRIMARY_CANT_HAVE_NULL [1578])

  Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error: 1172 SQLSTATE: 42000 (ER_TOO_MANY_ROWS [1578])

  Message: Result consisted of more than one row

- Error: 1173 SQLSTATE: 42000 (ER_REQUIRES_PRIMARY_KEY [1578])

  Message: This table type requires a primary key

- Error: 1174 SQLSTATE: HY000 (ER_NO_RAID_COMPILED [1578])

  Message: This version of MySQL is not compiled with RAID support

- Error: 1175 SQLSTATE: HY000 (ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE [1579])

  Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column

- Error: 1176 SQLSTATE: HY000 (ER_KEY_DOES_NOT_EXITS [1579])

  Message: Key '%s' doesn't exist in table '%s'

- Error: 1177 SQLSTATE: 42000 (ER_CHECK_NO_SUCH_TABLE [1579])

  Message: Can't open table

- Error: 1178 SQLSTATE: 42000 (ER_CHECK_NOT_IMPLEMENTED [1579])

  Message: The storage engine for the table doesn't support %s

- Error: 1179 SQLSTATE: 25000 (ER_CANT_DO_THIS_DURING_AN_TRANSACTION [1579])

  Message: You are not allowed to execute this command in a transaction

- Error: 1180 SQLSTATE: HY000 (ER_ERROR_DURING_COMMIT [1579])

  Message: Got error %d during COMMIT

- Error: 1181 SQLSTATE: HY000 (ER_ERROR_DURING_ROLLBACK [1579])

  Message: Got error %d during ROLLBACK

- Error: 1182 SQLSTATE: HY000 (ER_ERROR_DURING_FLUSH_LOGS [1579])

  Message: Got error %d during FLUSH_LOGS

- Error: 1183 SQLSTATE: HY000 (ER_ERROR_DURING_CHECKPOINT [1579])

  Message: Got error %d during CHECKPOINT

- Error: 1184 SQLSTATE: 08S01 (ER_NEW_ABORTING_CONNECTION [1579])

  Message: Aborted connection %ld to db: '%s' user: '%s' host: `%s' (%s)

- Error: 1185 SQLSTATE: HY000 (ER_DUMP_NOT_IMPLEMENTED [1579])

  Message: The storage engine for the table does not support binary table dump

- Error: 1186 SQLSTATE: HY000 (ER_FLUSH_MASTER_BINLOG_CLOSED [1579])

  Message: Binlog closed, cannot RESET MASTER

- Error: 1187 SQLSTATE: HY000 (ER_INDEX_REBUILD [1579])

  Message: Failed rebuilding the index of dumped table '%s'

- Error: 1188 SQLSTATE: HY000 (ER_MASTER [1579])

  Message: Error from master: '%s'

- Error: 1189 SQLSTATE: 08S01 (ER_MASTER_NET_READ [1579])

  Message: Net error reading from master

- Error: 1190 SQLSTATE: 08S01 (ER_MASTER_NET_WRITE [1580])

  Message: Net error writing to master

- Error: 1191 SQLSTATE: HY000 (ER_FT_MATCHING_KEY_NOT_FOUND [1580])

  Message: Can't find FULLTEXT index matching the column list

- Error: 1192 SQLSTATE: HY000 (ER_LOCK_OR_ACTIVE_TRANSACTION [1580])

  Message: Can't execute the given command because you have active locked tables or an active transaction

- Error: 1193 SQLSTATE: HY000 (ER_UNKNOWN_SYSTEM_VARIABLE [1580])

  Message: Unknown system variable '%s'

- Error: 1194 SQLSTATE: HY000 (ER_CRASHED_ON_USAGE [1580])

  Message: Table '%s' is marked as crashed and should be repaired

- Error: 1195 SQLSTATE: HY000 (ER_CRASHED_ON_REPAIR [1580])

  Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: 1196 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK [1580])

  Message: Some non-transactional changed tables couldn't be rolled back

- Error: 1197 SQLSTATE: HY000 (ER_TRANS_CACHE_FULL [1580])

  Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: 1198 SQLSTATE: HY000 (ER_SLAVE_MUST_STOP [1580])

  Message: This operation cannot be performed with a running slave; run STOP SLAVE first

- Error: 1199 SQLSTATE: HY000 (ER_SLAVE_NOT_RUNNING [1580])

  Message: This operation requires a running slave; configure slave and do START SLAVE

- Error: 1200 SQLSTATE: HY000 (ER_BAD_SLAVE [1580])

  Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error: 1201 SQLSTATE: HY000 (ER_MASTER_INFO [1580])

  Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: 1202 SQLSTATE: HY000 (ER_SLAVE_THREAD [1580])

  Message: Could not create slave thread; check system resources

- Error: 1203 SQLSTATE: 42000 (ER_TOO_MANY_USER_CONNECTIONS [1580])

  Message: User %s has already more than 'max_user_connections' active connections

- Error: 1204 SQLSTATE: HY000 (ER_SET_CONSTANTS_ONLY [1580])

Message: You may only use constant expressions with SET

- Error: 1205 SQLSTATE: HY000 (ER_LOCK_WAIT_TIMEOUT [1581])

  Message: Lock wait timeout exceeded; try restarting transaction

- Error: 1206 SQLSTATE: HY000 (ER_LOCK_TABLE_FULL [1581])

  Message: The total number of locks exceeds the lock table size

- Error: 1207 SQLSTATE: 25000 (ER_READ_ONLY_TRANSACTION [1581])

  Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: 1208 SQLSTATE: HY000 (ER_DROP_DB_WITH_READ_LOCK [1581])

  Message: DROP DATABASE not allowed while thread is holding global read lock

- Error: 1209 SQLSTATE: HY000 (ER_CREATE_DB_WITH_READ_LOCK [1581])

  Message: CREATE DATABASE not allowed while thread is holding global read lock

- Error: 1210 SQLSTATE: HY000 (ER_WRONG_ARGUMENTS [1581])

  Message: Incorrect arguments to %s

- Error: 1211 SQLSTATE: 42000 (ER_NO_PERMISSION_TO_CREATE_USER [1581])

  Message: '%s'@'%s' is not allowed to create new users

- Error: 1212 SQLSTATE: HY000 (ER_UNION_TABLES_IN_DIFFERENT_DIR [1581])

  Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: 1213 SQLSTATE: 40001 (ER_LOCK_DEADLOCK [1581])

  Message: Deadlock found when trying to get lock; try restarting transaction

- Error: 1214 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_FT [1581])

  Message: The used table type doesn't support FULLTEXT indexes

- Error: 1215 SQLSTATE: HY000 (ER_CANNOT_ADD_FOREIGN [1581])

  Message: Cannot add foreign key constraint

- Error: 1216 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW [1581])

  Message: Cannot add or update a child row: a foreign key constraint fails

- Error: 1217 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED [1581])

  Message: Cannot delete or update a parent row: a foreign key constraint fails

- Error: 1218 SQLSTATE: 08S01 (ER_CONNECT_TO_MASTER [1581])

  Message: Error connecting to master: %s

- Error: 1219 SQLSTATE: HY000 (ER_QUERY_ON_MASTER [1581])

Message: Error running query on master: %s

- Error: 1220 SQLSTATE: HY000 (ER_ERROR_WHEN_EXECUTING_COMMAND [1582])

  Message: Error when executing command %s: %s

- Error: 1221 SQLSTATE: HY000 (ER_WRONG_USAGE [1582])

  Message: Incorrect usage of %s and %s

- Error: 1222 SQLSTATE: 21000 (ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT [1582])

  Message: The used SELECT statements have a different number of columns

- Error: 1223 SQLSTATE: HY000 (ER_CANT_UPDATE_WITH_READLOCK [1582])

  Message: Can't execute the query because you have a conflicting read lock

- Error: 1224 SQLSTATE: HY000 (ER_MIXING_NOT_ALLOWED [1582])

  Message: Mixing of transactional and non-transactional tables is disabled

- Error: 1225 SQLSTATE: HY000 (ER_DUP_ARGUMENT [1582])

  Message: Option '%s' used twice in statement

- Error: 1226 SQLSTATE: 42000 (ER_USER_LIMIT_REACHED [1582])

  Message: User '%s' has exceeded the '%s' resource (current value: %ld)

- Error: 1227 SQLSTATE: HY000 (ER_SPECIFIC_ACCESS_DENIED_ERROR [1582])

  Message: Access denied; you need the %s privilege for this operation

- Error: 1228 SQLSTATE: HY000 (ER_LOCAL_VARIABLE [1582])

  Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: 1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE [1582])

  Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL

- Error: 1230 SQLSTATE: 42000 (ER_NO_DEFAULT [1582])

  Message: Variable '%s' doesn't have a default value

- Error: 1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR [1582])

  Message: Variable '%s' can't be set to the value of '%s'

- Error: 1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR [1582])

  Message: Incorrect argument type to variable '%s'

- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ [1582])

  Message: Variable '%s' can only be set, not read

- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE [1582])

Message: Incorrect usage/placement of '%s'

- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET [1583])

  Message: This version of MySQL doesn't yet support '%s'

- Error: 1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG [1583])

  Message: Got fatal error %d: '%s' from master when reading data from binary log

- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE [1583])

  Message: Slave SQL thread ignored the query because of replicate-*-table rules

- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR [1583])

  Message: Variable '%s' is a %s variable

- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF [1583])

  Message: Incorrect foreign key definition for '%s': %s

- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF [1583])

  Message: Key reference and table reference don't match

- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS [1583])

  Message: Operand should contain %d column(s)

- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW [1583])

  Message: Subquery returns more than 1 row

- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER [1583])

  Message: Unknown prepared statement handler (%.*s) given to %s

- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB [1583])

  Message: Help database is corrupt or does not exist

- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE [1583])

  Message: Cyclic reference on subqueries

- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT [1583])

  Message: Converting column '%s' from %s to %s

- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE [1583])

  Message: Reference '%s' not supported (%s)

- Error: 1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS [1583])

  Message: Every derived table must have its own alias

- Error: 1249 SQLSTATE: 01000 (ER_SELECT_REDUCED [1583])

Message: Select %u was reduced during optimization

- Error: 1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE [1584])

  Message: Table '%s' from one of the SELECTs cannot be used in %s

- Error: 1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE [1584])

  Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error: 1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL [1584])

  Message: All parts of a SPATIAL index must be NOT NULL

- Error: 1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH [1584])

  Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING [1584])

  Message: Slave is already running

- Error: 1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING [1584])

  Message: Slave has already been stopped

- Error: 1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS [1584])

  Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error: 1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR [1584])

  Message: ZLIB: Not enough memory

- Error: 1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR [1584])

  Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: 1259 SQLSTATE: HY000 (ER_ZLIB_Z_DATA_ERROR [1584])

  Message: ZLIB: Input data corrupted

- Error: 1260 SQLSTATE: HY000 (ER_CUT_VALUE_GROUP_CONCAT [1584])

  Message: %d line(s) were cut by GROUP_CONCAT()

- Error: 1261 SQLSTATE: 01000 (ER_WARN_TOO_FEW_RECORDS [1584])

  Message: Row %ld doesn't contain data for all columns

- Error: 1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS [1584])

  Message: Row %ld was truncated; it contained more data than there were input columns

- Error: 1263 SQLSTATE: 01000 (ER_WARN_NULL_TO_NOTNULL [1584])

  Message: Data truncated; NULL supplied to NOT NULL column '%s' at row %ld

- Error: 1264 SQLSTATE: 01000 (ER_WARN_DATA_OUT_OF_RANGE [1585])

  Message: Data truncated; out of range for column '%s' at row %ld

- Error: 1265 SQLSTATE: 01000 (ER_WARN_DATA_TRUNCATED [1585])

  Message: Data truncated for column '%s' at row %ld

- Error: 1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER [1585])

  Message: Using storage engine %s for table '%s'

- Error: 1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS [1585])

  Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error: 1268 SQLSTATE: HY000 (ER_DROP_USER [1585])

  Message: Can't drop one or more of the requested users

- Error: 1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS [1585])

  Message: Can't revoke all privileges, grant for one or more of the requested users

- Error: 1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS [1585])

  Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error: 1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS [1585])

  Message: Illegal mix of collations for operation '%s'

- Error: 1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT [1585])

  Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION [1585])

  Message: Unknown collation: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS [1585])

  Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE [1585])

  Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED [1585])

  Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d

- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND [1585])

  Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL

- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE [1585])

Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart

- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED [1586])

  Message: SQL thread is not to be started so UNTIL options are ignored

- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX [1586])

  Message: Incorrect index name '%s'

- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG [1586])

  Message: Incorrect catalog name '%s'

- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE [1586])

  Message: Query cache failed to set size %lu; new query cache size is %lu

- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN [1586])

  Message: Column '%s' cannot be part of FULLTEXT index

- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE [1586])

  Message: Unknown key cache '%s'

- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK [1586])

  Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work

- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE [1586])

  Message: Unknown table engine '%s'

- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX [1586])

  Message: '%s' is deprecated; use '%s' instead

- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE [1586])

  Message: The target table %s of the %s is not updatable

- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED [1586])

  Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working

- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT [1586])

  Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE [1586])

  Message: Column '%s' has duplicated value '%s' in %s

- Error: 1292 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE [1586])

  Message: Truncated incorrect %s value: '%s'

- Error: 1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS [1587])

  Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: 1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE [1587])

  Message: Invalid ON UPDATE clause for '%s' column

- Error: 1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS [1587])

  Message: This command is not supported in the prepared statement protocol yet

- Error: 1296 SQLSTATE: HY000 (ER_GET_ERRMSG [1587])

  Message: Got error %d '%s' from %s

- Error: 1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG [1587])

  Message: Got temporary error %d '%s' from %s

- Error: 1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE [1587])

  Message: Unknown or incorrect time zone: '%s'

- Error: 1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP [1587])

  Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: 1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING [1587])

  Message: Invalid %s character string: '%s'

- Error: 1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED [1587])

  Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: 1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS [1587])

  Message: Conflicting declarations: '%s%s' and '%s%s'

B.4 Client Error Codes and Messages

Client error information comes from the following source files:

- The Error values and the symbols in parentheses correspond to definitions in the include/errmsg.h MySQL source file.
- The Message values correspond to the error messages that are listed in the libmysql/errmsg.c file. %d and %s represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: 2000 (CR_UNKNOWN_ERROR [1587])

  Message: Unknown MySQL error

- Error: 2001 (CR_SOCKET_CREATE_ERROR [1587])

Message: Can't create UNIX socket (%d)

- Error: 2002 (CR_CONNECTION_ERROR [1588])

  Message: Can't connect to local MySQL server through socket '%s' (%d)

- Error: 2003 (CR_CONN_HOST_ERROR [1588])

  Message: Can't connect to MySQL server on '%s' (%d)

- Error: 2004 (CR_IPSOCK_ERROR [1588])

  Message: Can't create TCP/IP socket (%d)

- Error: 2005 (CR_UNKNOWN_HOST [1588])

  Message: Unknown MySQL server host '%s' (%d)

- Error: 2006 (CR_SERVER_GONE_ERROR [1588])

  Message: MySQL server has gone away

- Error: 2007 (CR_VERSION_ERROR [1588])

  Message: Protocol mismatch; server version = %d, client version = %d

- Error: 2008 (CR_OUT_OF_MEMORY [1588])

  Message: MySQL client ran out of memory

- Error: 2009 (CR_WRONG_HOST_INFO [1588])

  Message: Wrong host info

- Error: 2010 (CR_LOCALHOST_CONNECTION [1588])

  Message: Localhost via UNIX socket

- Error: 2011 (CR_TCP_CONNECTION [1588])

  Message: %s via TCP/IP

- Error: 2012 (CR_SERVER_HANDSHAKE_ERR [1588])

  Message: Error in server handshake

- Error: 2013 (CR_SERVER_LOST [1588])

  Message: Lost connection to MySQL server during query

- Error: 2014 (CR_COMMANDS_OUT_OF_SYNC [1588])

  Message: Commands out of sync; you can't run this command now

- Error: 2015 (CR_NAMEDPIPE_CONNECTION [1588])

  Message: Named pipe: %s

- Error: 2016 (CR_NAMEDPIPEWAIT_ERROR [1588])

Message: Can't wait for named pipe to host: %s pipe: %s (%lu)

- Error: 2017 (CR_NAMEDPIPEOPEN_ERROR [1589])

  Message: Can't open named pipe to host: %s pipe: %s (%lu)

- Error: 2018 (CR_NAMEDPIPESETSTATE_ERROR [1589])

  Message: Can't set state of named pipe to host: %s pipe: %s (%lu)

- Error: 2019 (CR_CANT_READ_CHARSET [1589])

  Message: Can't initialize character set %s (path: %s)

- Error: 2020 (CR_NET_PACKET_TOO_LARGE [1589])

  Message: Got packet bigger than 'max_allowed_packet' bytes

- Error: 2021 (CR_EMBEDDED_CONNECTION [1589])

  Message: Embedded server

- Error: 2022 (CR_PROBE_SLAVE_STATUS [1589])

  Message: Error on SHOW SLAVE STATUS:

- Error: 2023 (CR_PROBE_SLAVE_HOSTS [1589])

  Message: Error on SHOW SLAVE HOSTS:

- Error: 2024 (CR_PROBE_SLAVE_CONNECT [1589])

  Message: Error connecting to slave:

- Error: 2025 (CR_PROBE_MASTER_CONNECT [1589])

  Message: Error connecting to master:

- Error: 2026 (CR_SSL_CONNECTION_ERROR [1589])

  Message: SSL connection error

- Error: 2027 (CR_MALFORMED_PACKET [1589])

  Message: Malformed packet

- Error: 2028 (CR_WRONG_LICENSE [1589])

  Message: This client library is licensed only for use with MySQL servers having '%s' license

- Error: 2029 (CR_NULL_POINTER [1589])

  Message: Invalid use of null pointer

- Error: 2030 (CR_NO_PREPARE_STMT [1589])

  Message: Statement not prepared

- Error: 2031 (CR_PARAMS_NOT_BOUND [1589])

Message: No data supplied for parameters in prepared statement

- Error: 2032 (CR_DATA_TRUNCATED [1590])

  Message: Data truncated

- Error: 2033 (CR_NO_PARAMETERS_EXISTS [1590])

  Message: No parameters exist in the statement

- Error: 2034 (CR_INVALID_PARAMETER_NO [1590])

  Message: Invalid parameter number

- Error: 2035 (CR_INVALID_BUFFER_USE [1590])

  Message: Can't send long data for non-string/non-binary data types (parameter: %d)

- Error: 2036 (CR_UNSUPPORTED_PARAM_TYPE [1590])

  Message: Using unsupported buffer type: %d (parameter: %d)

- Error: 2037 (CR_SHARED_MEMORY_CONNECTION [1590])

  Message: Shared memory: %s

- Error: 2038 (CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR [1590])

  Message: Can't open shared memory; client could not create request event (%lu)

- Error: 2039 (CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR [1590])

  Message: Can't open shared memory; no answer event received from server (%lu)

- Error: 2040 (CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR [1590])

  Message: Can't open shared memory; server could not allocate file mapping (%lu)

- Error: 2041 (CR_SHARED_MEMORY_CONNECT_MAP_ERROR [1590])

  Message: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error: 2042 (CR_SHARED_MEMORY_FILE_MAP_ERROR [1590])

  Message: Can't open shared memory; client could not allocate file mapping (%lu)

- Error: 2043 (CR_SHARED_MEMORY_MAP_ERROR [1590])

  Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: 2044 (CR_SHARED_MEMORY_EVENT_ERROR [1590])

  Message: Can't open shared memory; client could not create %s event (%lu)

- Error: 2045 (CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR [1590])

  Message: Can't open shared memory; no answer from server (%lu)

- Error: 2046 (CR_SHARED_MEMORY_CONNECT_SET_ERROR [1590])

Message: Can't open shared memory; cannot send request event to server (%lu)

- Error: `2047` (`CR_CONN_UNKNOW_PROTOCOL` [1591])

  Message: Wrong or unknown protocol

- Error: `2048` (`CR_INVALID_CONN_HANDLE` [1591])

  Message: Invalid connection handle

- Error: `2049` (`CR_SECURE_AUTH` [1591])

  Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error: `2050` (`CR_FETCH_CANCELED` [1591])

  Message: Row retrieval was canceled by mysql_stmt_close() call

- Error: `2051` (`CR_NO_DATA` [1591])

  Message: Attempt to read column without prior row fetch

- Error: `2052` (`CR_NO_STMT_METADATA` [1591])

  Message: Prepared statement contains no metadata

B.5 Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

B.5.1 How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:

  - The keyboard does not work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light does not change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
  - The mouse pointer does not move.
  - The machine does not answer to a remote machine's pings.
  - Other programs that are not related to MySQL do not behave correctly.
  - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

  In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as `glibc`) are up to date.

It is always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See Section 5.3, “MySQL Server Logs”.
- If you do not think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it does not want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it is time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the “copy and paste” method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only “the system does not work.” This provides us with no information about what could be the problem.

If a program fails, it is always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in Section 1.8, “How to Report Bugs or Problems”.

B.5.2 Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

B.5.2.1 `Access denied`

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server permits client programs to use when connecting. See Section 5.5, “The MySQL Access Privilege System”, and Section 5.5.7, “Causes of Access-Denied Errors”.

B.5.2.2 Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the mysqld server in two different ways: By using a Unix socket file to connect through a file in the file system (default /tmp/mysql.sock), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you don't specify a host name or if you specify the special host name localhost.

If the MySQL server is running on Windows 9x or Me, you can connect only using TCP/IP. If the server is running on Windows NT, 2000, XP, or 2003 and is started with the --enable-named-pipe [386] option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is MySQL by default. If you don't give a host name when connecting to mysqld, a MySQL client first tries to connect to the named pipe. If that doesn't work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using . as the host name.

The error (2002) Can't connect to ... normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket file name or TCP/IP port number when trying to connect to the server. You should also check that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

The error (2003) Can't connect to MySQL server on '*server*' (10061) indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, and that the network port you specified is the one configured on the server.

Start by checking whether there is a process named mysqld running on your server host. (Use ps xa | grep mysqld on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”.

If a mysqld process is running, you can check it by trying the following commands. The port number or Unix socket file name might be different in your setup. host_ip represents the IP address of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotation marks with the hostname command; these cause the output of hostname (that is, the current host name) to be substituted into the mysqladmin command. If you have no hostname command or are running on Windows, you can manually type the host name of your machine (without backticks) following the -h option. You can also try -h 127.0.0.1 to connect with TCP/IP to the local host.

Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with --skip-networking [393], it will not accept TCP/IP connections at all. If the server was started with --bind-address=127.0.0.1 [384], it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running on the local host. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: `Connection to MySQL Server Failing on Windows`.
- You are running on a system that uses MIT-pthreads. If you are running on a system that doesn't have native threads, `mysqld` uses the MIT-pthreads package. See Section 2.1.1, “Operating Systems On Which MySQL Is Known To Run”. However, not all MIT-pthreads versions support Unix socket files. On a system without socket file support, you must always specify the host name explicitly when connecting to the server. Try using this command to check the connection to the server:

  ```
  shell> mysqladmin -h `hostname` version
  ```

- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See Section B.5.4.5, “How to Protect or Change the MySQL Unix Socket File”.
- You have started the `mysqld` server with the `--socket=/path/to/socket` [394] option, but forgotten to tell client programs the new name of the socket file. If you change the socket path name for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` [227] option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

  ```
  shell> netstat -ln | grep mysql
  ```

  See Section B.5.4.5, “How to Protect or Change the MySQL Unix Socket File”.
- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill` or with the `mysql_zap` script) before you can restart the MySQL server. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` [394] option that specifies a socket file name in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` [423] variable.)

- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, make sure you have disabled SELinux protection for the `mysqld` process.

Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows does not allow for enough ephemeral (short-lived) ports to serve those connections.

The purpose of `TIME_WAIT` is to keep a connection accepting packets even after the connection has been closed. This is because Internet routing can cause a packet to take a slow route to its destination and it may arrive after both sides have agreed to close. If the port is in use for a new connection, that packet from the old connection could break the protocol or compromise personal information from the original connection. The `TIME_WAIT` delay prevents this by ensuring that the port cannot be reused until after some time has been permitted for those delayed packets to arrive.

It is safe to reduce `TIME_WAIT` greatly on LAN connections because there is little chance of packets arriving at very long delays, as they could through the Internet with its comparatively large distances and latencies.

By default, Windows allows 5000 ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a `TIME_WAIT` status for 120 seconds. The port will not be available again until this time expires.

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the `TIME_WAIT` status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

IMPORTANT: The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore the registry if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: http://support.microsoft.com/kb/256986/EN-US/.

1. Start Registry Editor (`Regedt32.exe`).
2. Locate the following key in the registry:

   ```
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
   ```

3. On the `Edit` menu, click `Add Value`, and then add the following registry value:

   ```
   Value Name: MaxUserPort
   Data Type: REG_DWORD
   Value: 65534
   ```

   This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the `Edit` menu, click `Add Value`, and then add the following registry value:

   ```
   Value Name: TcpTimedWaitDelay
   Data Type: REG_DWORD
   ```

```
Value: 30
```

This sets the number of seconds to hold a TCP port connection in `TIME_WAIT` state before closing. The valid range is between 0 (zero) and 300 (decimal). The default value is 0x78 (120 decimal).

5. Quit Registry Editor.

6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

B.5.2.3 Lost connection to MySQL server

There are three likely causes for this error message.

Usually it indicates network connectivity trouble and you should check the condition of your network if this error occurs frequently. If the error message includes “during query,” this is probably the case you are experiencing.

Sometimes the “during query” form happens when millions of rows are being sent as part of one or more queries. If you know that this is happening, you should try increasing `net_read_timeout` [422] from its default of 30 seconds to 60 seconds or longer, sufficient for the data transfer to complete.

More rarely, it can happen when the client is attempting the initial connection to the server. In this case, if your `connect_timeout` [409] value is set to only a few seconds, you may be able to resolve the problem by increasing it to ten seconds, perhaps more if you have a very long distance or slow connection. You can determine whether you are experiencing this more uncommon cause by using `SHOW STATUS LIKE 'Aborted_connects'`. It will increase by one for each initial connection attempt that the server aborts. You may see “reading authorization packet” as part of the error message; if so, that also suggests that this is the solution that you need.

If the cause is none of those just described, you may be experiencing a problem with `BLOB` values that are larger than `max_allowed_packet` [418], which can cause this error with some clients. Sometime you may see an `ER_NET_PACKET_TOO_LARGE` [1577] error, and that confirms that you need to increase `max_allowed_packet` [418].

B.5.2.4 Client does not support authentication protocol

MySQL 4.1 and up uses an authentication protocol based on a password hashing algorithm that is incompatible with that used by older clients. If you upgrade the server from 4.0, attempts to connect to it with an older client may fail with the following message:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

To solve this problem, you should use one of the following approaches:

- Upgrade all client programs to use a 4.1.1 or newer client library.
- When connecting to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the `SET PASSWORD` statement and the `OLD_PASSWORD()` [868] function:

```
mysql> SET PASSWORD FOR
```

```
    -> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
```

Alternatively, use `UPDATE` and `FLUSH PRIVILEGES`:

```
mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('newpwd')
    -> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

Substitute the password you want to use for "*newpwd*" in the preceding examples. MySQL cannot tell you what the original password was, so you'll need to pick a new one.

- Tell the server to use the older password hashing algorithm:

  1. Start `mysqld` with the `--old-passwords` [391] option.
  2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

     ```
     mysql> SELECT Host, User, Password FROM mysql.user
         -> WHERE LENGTH(Password) > 16;
     ```

     For each account record displayed by the query, use the `Host` and `User` values and assign a password using the `OLD_PASSWORD()` [868] function and either `SET PASSWORD` or `UPDATE`, as described earlier.

Note

In PHP, the standard `mysql` extension does not support the new authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the `mysql` extension with MySQL 4.1 or newer, you will need to follow one of the options discussed above for configuring MySQL to work with old clients. The `mysqli` extension (stands for "MySQL, Improved"; new in PHP 5) **is** compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this newer MySQL client library for PHP. For more information about the `mysqli` extension, see http://php.net/mysqli.

It may also be possible to compile the older `mysql` extension against the new MySQL client library. This is beyond the scope of this Manual; consult the PHP documentation for more information. You also be able to obtain assistance with these issues in our MySQL with PHP forum.

For additional background on password hashing and authentication, see Section 5.4.2.3, "Password Hashing in MySQL".

B.5.2.5 Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` [226] or `-p` option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it,

change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

B.5.2.6 Host '*host_name*' is blocked

If the following error occurs, it means that `mysqld` has received many connection requests from the given host that were interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The number of interrupted connect requests permitted is determined by the value of the `max_connect_errors` [419] system variable. After `max_connect_errors` [419] failed requests, `mysqld` assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you execute a `mysqladmin flush-hosts` command or issue a `FLUSH HOSTS` statement. See Section 5.1.3, “Server System Variables”.

By default, `mysqld` blocks a host after 10 connection errors. You can adjust the value by starting the server like this:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

If you get this error message for a given host, you should first verify that there isn't anything wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` [419] variable.

B.5.2.7 Too many connections

If you get a `Too many connections` error when you try to connect to the `mysqld` server, this means that all available connections are in use by other clients.

The number of connections permitted is controlled by the `max_connections` [419] system variable. Its default value is 100. If you need to support more connections, you should set a larger value for this variable.

`mysqld` actually permits `max_connections+1` [419] clients to connect. The extra connection is reserved for use by accounts that have the `SUPER` [493] privilege. By granting the `SUPER` [493] privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 12.4.5.19, “`SHOW PROCESSLIST` Syntax”.

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform, the amount of RAM available, how much RAM is used for each connection, the workload from each connection, and the desired response time. Linux or Solaris should be able to support at 500 to 1000 simultaneous connections routinely and as many as 10,000 connections if you have many gigabytes of RAM available and the workload from each is low or the response time target undemanding. Windows is limited to (open tables × 2 + open connections) < 2048 due to the Posix compatibility layer used on that platform.

Increasing `open-files-limit` [391] may be necessary. Also see Section 2.12.1.4, “Linux Postinstallation Notes”, for how to raise the operating system limit on how many handles can be used by MySQL.

B.5.2.8 Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` [263] option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

B.5.2.9 `MySQL server has gone away`

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent).

Error Code	Description
`CR_SERVER_GONE_ERROR` [1588]	The client couldn't send a question to the server.
`CR_SERVER_LOST` [1588]	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` [434] variable when you start `mysqld`. See Section 5.1.3, “Server System Variables”.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the `MySQL server has gone away` error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).
- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` [434] expired) before the command was issued.

  The problem on Windows is that in some cases MySQL doesn't get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

In this case, even if the `reconnect` flag in the `MYSQL` structure is equal to 1, MySQL does not automatically reconnect and re-issue the query as it doesn't know if the server did get the original query or not.

The solution to this is to either do a `mysql_ping()` on the connection if there has been a long time since the last query (this is what `MyODBC` does) or set `wait_timeout` [434] on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's `max_allowed_packet` [418] variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in Section B.5.2.10, “Packet Too Large”.

  An `INSERT` or `REPLACE` statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per `INSERT` or `REPLACE`.

- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.

- It is also possible to see this error if host name lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working—from MySQL's point of view the problem is indistinguishable from any other network timeout.

  You may also see the `MySQL server has gone away` error if MySQL is started with the `--skip-networking` [393] option.

  Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.

- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the server's uptime. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.

You can get more information about the lost connections by starting `mysqld` with the `--log-warnings=2` [389] option. This logs some of the disconnected errors in the `hostname.err` file. See Section 5.3.1, “The Error Log”.

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See Section B.5.4.2, “What to Do If MySQL Keeps Crashing”.

- If a specific query kills `mysqld` and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See Section 18.4, “Porting to Other Systems”.

- What is the value of the `wait_timeout` [434] system variable in the MySQL server? (`mysqladmin variables` gives you the value of this variable.)
- Have you tried to run `mysqld` with the general query log enabled to determine whether the problem query appears in the log? (See Section 5.3.2, “The General Query Log”.)

See also Section B.5.2.11, “Communication Errors and Aborted Connections”, and Section 1.8, “How to Report Bugs or Problems”.

B.5.2.10 Packet Too Large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a master replication server to a slave.

In MySQL 3.23, the largest possible packet is 16MB, due to limits in the client/server protocol. In MySQL 4.0.1 and up, the limit is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` [418] bytes, it issues an `ER_NET_PACKET_TOO_LARGE` [1577] error and closes the connection. With some clients, you may also get a `Lost connection to MySQL server during query` error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` [418] variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` [418] variable is 16MB. That is also the maximum value before MySQL 4.0. To set a larger value from 4.0 on, start `mysql` like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` [418] value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

Before MySQL 4.0, use this syntax instead:

```
shell> mysqld --set-variable=max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet` [418]. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=16M
```

Before MySQL 4.0, use this syntax instead:

```
[mysqld]
set-variable = max_allowed_packet=16M
```

It's safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets

between the client and server and also to ensure that you don't run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

B.5.2.11 Communication Errors and Aborted Connections

The server error log can be a useful source of information about connection problems. See Section 5.3.1, "The Error Log". Starting with MySQL 3.23.40, if you start the server with the `--warnings` [389] option (or `--log-warnings` [389] from MySQL 4.0.3 on), you might find messages like this in your error log:

```
010301 14:38:23  Aborted connection 854 to db: 'users' user: 'josh'
```

If a client successfully connects but later disconnects improperly or is terminated, the server increments the `Aborted_clients` [449] status variable, and logs an `Aborted connection` message to the error log. The cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` [434] or `interactive_timeout` [414] seconds without issuing any requests to the server. See Section 5.1.3, "Server System Variables".
- The client program ended abruptly in the middle of a data transfer.

If a client is unable even to connect, the server increments the `Aborted_connects` [449] status variable. Unsuccessful connection attempts can occur for the following reasons:

- A client does not have privileges to connect to a database.
- A client uses an incorrect password.
- A connection packet does not contain the right information.
- It takes more than `connect_timeout` [409] seconds to get a connect packet. See Section 5.1.3, "Server System Variables".

If these kinds of things happen, it might indicate that someone is trying to break into your server! Messages for these types of problems are logged to the general query log if it is enabled.

Other reasons for problems with aborted clients or aborted connections:

- The `max_allowed_packet` [418] variable value is too small or queries require more memory than you have allocated for `mysqld`. See Section B.5.2.10, "Packet Too Large".
- Use of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file using FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. Switch the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and test the results to determine the best setting.
- A problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.

See also Section B.5.2.9, “`MySQL server has gone away`”.

B.5.2.12 `The table is full`

The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

Operating System	File-size Limit
Win32 w/ FAT/FAT32	2GB/4GB
Win32 w/ NTFS	2TB (possibly larger)
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4+	(using ext3 file system) 4TB
Solaris 9/10	16TB
MacOS X w/ HFS+	2TB
NetWare w/NSS file system	8TB

Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

On Linux 2.2, you can get `MyISAM` tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 file system. Most current Linux distributions are based on kernel 2.4 or higher and include all the required LFS patches. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). With JFS and XFS, petabyte and larger files are possible on Linux.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's *Large File Support in Linux* page at http://www.suse.de/~aj/linux_lfs.html.

If you do encounter a full-table error, there are several reasons why it might have occurred:

- You are using a MySQL server older than 3.23 and an in-memory temporary table becomes larger than `tmp_table_size` [432] bytes. To avoid this problem, you can use the `--tmp_table_size=val` option to make `mysqld` increase the temporary table size or use the SQL option `sql_big_tables` before you issue the problematic query. See Section 12.4.4, “`SET` Syntax”.

  Under Windows you may get an error `OS error code 22: Invalid argument` if you are using NTFS file system compression and the size od your data or temporary table files exceesds 30GB. This is due to a limitation in the NTFS file system that it is unable to compress files of this size. See http://msdn2.microsoft.com/en-us/library/aa364219.aspx.

  You can also start `mysqld` with the `--big-tables` [384] option. This is exactly the same as using `sql_big_tables` for all queries.

  As of MySQL 3.23, this problem should not occur. If an in-memory temporary table becomes larger than `tmp_table_size` [432], the server automatically converts it to a disk-based `MyISAM` table.

- The `InnoDB` storage engine maintains `InnoDB` tables within a tablespace that can be created from several files. This enables a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which permits extremely large tables. The maximum tablespace size is 64TB.

  If you are using `InnoDB` tables and run out of room in the `InnoDB` tablespace. In this case, the solution is to extend the `InnoDB` tablespace. See Section 13.2.6, “Adding, Removing, or Resizing `InnoDB` Data and Log Files”.

- You are using `ISAM` or `MyISAM` tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a `MyISAM` table and the space required for the table exceeds what is permitted by the internal pointer size. `MyISAM` creates tables to permit up to 4GB by default, but this limit can be changed up to the maximum permissible size of 65,536TB (256^7 – 1 bytes).

  If you need a `MyISAM` table that is larger than the default limit and your operating system supports large files, the `CREATE TABLE` statement supports `AVG_ROW_LENGTH` and `MAX_ROWS` options. See Section 12.1.5, “`CREATE TABLE` Syntax”. The server uses these options to determine how large a table to permit.

  If the pointer size is too small for an existing table, you can change the options with `ALTER TABLE` to increase a table's maximum permissible size. See Section 12.1.2, “`ALTER TABLE` Syntax”.

  ```
  ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
  ```

  You have to specify `AVG_ROW_LENGTH` only for tables with `BLOB` or `TEXT` columns; in this case, MySQL can't optimize the space required based only on the number of rows.

  To change the default size limit for `MyISAM` data and index table files, set the `myisam_data_pointer_size` [421], which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new tables if you do not specify the `MAX_ROWS` option. The value of `myisam_data_pointer_size` [421] can be from 2 to 7. A value of 4 permits table files up to 4GB; a value of 6 permits table files up to 256TB.

  You can check the maximum data and index sizes by using this statement:

  ```
  SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
  ```

  You also can use `myisamchk -dv /path/to/table-index-file`. See Section 12.4.5, “`SHOW` Syntax”, or Section 4.6.2, “`myisamchk` — MyISAM Table-Maintenance Utility”.

  Other ways to work around file-size limits for `MyISAM` tables are as follows:

  - If your large table is read only, you can use `myisampack` to compress it. `myisampack` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `myisampack` also can merge multiple tables into a single table. See Section 4.6.4, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.

    You can also get around the operating system file limit for `MyISAM` data files by using the `RAID` options for `CREATE TABLE`. See Section 12.1.5, “`CREATE TABLE` Syntax”.

  - MySQL includes a `MERGE` library that enables you to handle a collection of `MyISAM` tables that have identical structure as a single `MERGE` table. See Section 13.3, “The `MERGE` Storage Engine”.

- You are using the `NDB` storage engine, in which case you need to increase the values for the `DataMemory` and `IndexMemory` configuration parameters in your `config.ini` file. See Section 15.3.3.1, “MySQL Cluster Data Node Configuration Parameters”.
- You are using the `MEMORY` (`HEAP`) storage engine; in this case you need to increase the value of the `max_heap_table_size` [419] system variable. See Section 5.1.3, “Server System Variables”.

B.5.2.13 `Can't create/write to file`

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir` [394] option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See Section 4.2.3.3, “Using Option Files”.

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` [432] directory.

Check also the error code that you get with `perror`. One reason the server cannot write to a table is that the file system is full:

```
shell> perror 28
OS error code  28:  No space left on device
```

If you get an error of the following type during startup, it indicates that the file system or directory used for storing data files is write protected. Providing the write error is to a test file, this error is not serious and can be safely ignored.

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

B.5.2.14 `Commands out of sync`

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

B.5.2.15 `Ignoring user`

If you get the following error, it means that when `mysqld` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

`Found wrong password for user '`*`some_user`*`'@'`*`some_host`*`'; ignoring user`

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of `mysqld` with an old `user` table. You can check this by executing `mysqlshow mysql user` to see whether the `Password` column is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.

- The account has an old password (eight characters long) and you didn't start `mysqld` with the `--old-protocol` [391] option. Update the account in the `user` table to have a new password or restart `mysqld` with the `--old-protocol` [391] option.
- You have specified a password in the `user` table without using the `PASSWORD()` [868] function. Use `mysql` to update the account in the `user` table with a new password, making sure to use the `PASSWORD()` [868] function:

```
mysql> UPDATE user SET Password=PASSWORD('newpwd')
    -> WHERE User='some_user' AND Host='some_host';
```

B.5.2.16 `Table 'tbl_name' doesn't exist`

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a file system that has case-sensitive file names.
- Even for file systems that are not case sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See Section 12.4.5, “SHOW Syntax”.

B.5.2.17 `Can't initialize character set`

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multi-byte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running `configure` with the `--with-charset=charset_name` [97] or `--with-extra-charsets=charset_name` [97] option. See Section 2.9.3, “MySQL Source-Configuration Options”.

  All standard MySQL binaries are compiled with `--with-extra-charsets=complex` [97], which enables support for all multi-byte character sets. See Section 2.9.3, “MySQL Source-Configuration Options”.
- The character set is a simple character set that is not compiled into `mysqld`, and the character set definition files are not in the place where the client expects to find them.

  In this case, you need to use one of the following methods to solve the problem:

  - Recompile the client with support for the character set. See Section 2.9.3, “MySQL Source-Configuration Options”.
  - Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.

- Copy the character definition files to the path where the client expects them to be.

B.5.2.18 `'File' Not Found` and Similar Errors

If you get `ERROR '...' not found (errno: 23)`, `Can't open file: ... (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you haven't allocated enough file descriptors for the MySQL server. You can use the `perror` utility to get a description of what the error number means:

```
shell> perror 23
OS error code  23:  File table overflow
shell> perror 24
OS error code  24:  Too many open files
shell> perror 11
OS error code  11:  Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_cache` [431] system variable (the default value is 64). This may not entirely prevent running out of file descriptors because in some circumstances the server may attempt to extend the cache size temporarily, as described in Section 7.7.2, “How MySQL Opens and Closes Tables”. Reducing the value of `max_connections` [419] also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` [245] option to `mysqld_safe` or (as of MySQL 3.23.30) set the `open_files_limit` [423] system variable. See Section 5.1.3, “Server System Variables”. The easiest way to set these values is to add an option to your option file. See Section 4.2.3.3, “Using Option Files”. If you have an old version of `mysqld` that doesn't support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the “`#`” character to uncomment this line, and change the number `256` to set the number of file descriptors to be made available to `mysqld`.

`--open-files-limit` [245] and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a “hard” limit that can be overridden only if you start `mysqld_safe` or `mysqld` as `root` (just remember that you also need to start the server with the `--user` [395] option in this case so that it does not continue to run as `root` after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note

If you run the `tcsh` shell, `ulimit` does not work! `tcsh` also reports incorrect values when you ask for the current limits. In this case, you should start `mysqld_safe` using `sh`.

B.5.2.19 Table-Corruption Issues

If you have started `mysqld` with `--myisam-recover` [390], MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further.

See also Section 5.1.2, “Server Command Options”, and Section 18.4.1.6, “Making a Test Case If You Experience Table Corruption”.

B.5.3 Installation-Related Issues

B.5.3.1 Problems with File Permissions

If you have problems with file permissions, the `UMASK` environment variable might be set incorrectly when `mysqld` starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default `UMASK` value is `0660`. You can change this behavior by starting `mysqld_safe` as follows:

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database and `RAID` directories with an access permission value of `0700`. You can modify this behavior by setting the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, if you want to give group access to all new directories, you can do this:

```
shell> UMASK_DIR=504  # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

In MySQL 3.23.25 and above, MySQL assumes that the value for `UMASK` or `UMASK_DIR` is in octal if it starts with a zero.

See Section 2.13, “Environment Variables”.

B.5.4 Administration-Related Issues

B.5.4.1 How to Reset the Root Password

If you have never set a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, this is insecure. For instructions on assigning passwords, see Section 2.10.3, “Securing the Initial MySQL Accounts”.

If you know the `root` password, but want to change it, see Section 12.4.1.4, “`SET PASSWORD` Syntax”.

If you set a `root` password previously, but have forgotten it, you can set a new password. The following sections provide instructions for Windows and Unix systems, as well as generic instructions that apply to any system.

Resetting the Root Password: Windows Systems

On Windows, use the following procedure to reset the password for all MySQL `root` accounts:

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager: From the Start menu, select Control Panel, then Administrative Tools, then Services. Find the MySQL service in the list and stop it.

   If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Create a text file containing the following statements. Replace the password with the password that you want to use.

   ```
   UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
   FLUSH PRIVILEGES;
   ```

   Write the `UPDATE` and `FLUSH` statements each on a single line. The `UPDATE` statement resets the password for all `root` accounts, and the `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

4. Save the file. For this example, the file will be named `C:\mysql-init.txt`.

5. Open a console window to get to the command prompt: From the Start menu, select Run, then enter `cmd` as the command to be run.

6. Start the MySQL server with the special `--init-file` [387] option (notice that the backslash in the option value is doubled):

   ```
   C:\> C:\mysql\bin\mysqld-nt --init-file=C:\\mysql-init.txt
   ```

   If you installed MySQL to a location other than `C:\mysql`, adjust the command accordingly.

   The server executes the contents of the file named by the `--init-file` [387] option at startup, changing each `root` account password.

   You can also add the `--console` [385] option to the command if you want server output to appear in the console window rather than in a log file.

   Users of MySQL 4.1 and higher who installed MySQL using the MySQL Installation Wizard may need to specify a `--defaults-file` [235] option:

   ```
   C:\> "C:\Program Files\MySQL\MySQL Server 4.1\bin\mysqld-nt.exe"
           --defaults-file="C:\\Program Files\\MySQL\\MySQL Server 4.1\\my.ini"
           --init-file=C:\\mysql-init.txt
   ```

   The appropriate `--defaults-file` [235] setting can be found using the Services Manager: From the Start menu, select Control Panel, then Administrative Tools, then Services. Find the MySQL service in the list, right-click it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` [235] setting.

7. After the server has started successfully, delete `C:\mysql-init.txt`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the MySQL server, then restart it in normal mode again. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

Resetting the Root Password: Unix Systems

On Unix, use the following procedure to reset the password for all MySQL `root` accounts. The instructions assume that you will start the server so that it runs using the Unix login account that you normally use for running the server. For example, if you run the server using the `mysql` login account, you should log in as `mysql` before using the instructions. Alternatively, you can log in as `root`, but in this case you *must* start `mysqld` with the `--user=mysql` [395] option. If you start the server as `root` without using `--user=mysql` [395], the server may create `root`-owned files in the data directory, such as log files, and these may cause permission-related problems for future server startups. If that happens, you will need to either change the ownership of the files to `mysql` or remove them.

1. Log on to your system as the Unix user that the `mysqld` server runs as (for example, `mysql`).
2. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, host name, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the file name has an extension of `.pid` and begins with either `mysqld` or your system's host name.

   You can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process, using the path name of the `.pid` file in the following command:

   ```
   shell> kill `cat /mysql-data-directory/host_name.pid`
   ```

   Use backticks (not forward quotation marks) with the `cat` command. These cause the output of `cat` to be substituted into the `kill` command.
3. Create a text file containing the following statements. Replace the password with the password that you want to use.

   ```
   UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
   FLUSH PRIVILEGES;
   ```

   Write the `UPDATE` and `FLUSH` statements each on a single line. The `UPDATE` statement resets the password for all `root` accounts, and the `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.
4. Save the file. For this example, the file will be named `/home/me/mysql-init`. The file contains the password, so it should not be saved where it can be read by other users.
5. Start the MySQL server with the special `--init-file` [387] option:

   ```
   shell> mysqld_safe --init-file=/home/me/mysql-init &
   ```

   The server executes the contents of the file named by the `--init-file` [387] option at startup, changing each `root` account password.
6. After the server has started successfully, delete `/home/me/mysql-init`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally.

Resetting the Root Password: Generic Instructions

The preceding sections provide password-resetting instructions for Windows and Unix systems. Alternatively, on any platform, you can set the new password using the `mysql` client (but this approach is less secure):

1. Stop `mysqld` and restart it with the `--skip-grant-tables` [392] option. This enables anyone to connect without a password and with all privileges. If you normally start the server with the `--old-passwords` [391] option, include that option as well.
2. Connect to the `mysqld` server with this command:

   ```
   shell> mysql
   ```
3. Issue the following statements in the `mysql` client. Replace the password with the password that you want to use.

```
mysql> UPDATE mysql.user SET Password=PASSWORD('MyNewPass')
    ->                   WHERE User='root';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally (without the `--skip-grant-tables` [392] option).

B.5.4.2 What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See Section 5.3.1, “The Error Log”.

On some systems, you can find in the error log a stack trace of where `mysqld` died that you can resolve with the `resolve_stack_dump` program. See Section 18.4, “Porting to Other Systems”. Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with `--delay-key-write` [386], in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` [387] option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that does not support good file system locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all `MyISAM` tables, and restart `mysqld`. This ensures that you are running from a clean state. See Chapter 5, *MySQL Server Administration*.

- Start `mysqld` with the general query log enabled (see Section 5.3.2, “The General Query Log”). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.3.2, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have been able to locate the bug and should submit a bug report for it. See Section 1.8, “How to Report Bugs or Problems”.

- Try to make a test case that we can use to repeat the problem. See Section 18.4, “Porting to Other Systems”.

- Try running the tests in the `mysql-test` directory and the MySQL benchmarks. See Section 18.1.2, “The MySQL Test Suite”. They should test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks can be found in the `sql-bench` directory in a source distribution or, for a binary distribution, in the `sql-bench` directory under your MySQL installation directory.

- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)

- If you configure MySQL for debugging, it is much easier to gather information about possible errors if something goes wrong. Configuring MySQL for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening. Reconfigure MySQL with the `--with-debug` [98] or `--with-debug=full` [98] option to `configure` and then recompile. See Section 18.4, “Porting to Other Systems”.

- Make sure that you have applied the latest patches for your operating system.

- Use the `--skip-external-locking` [392] option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` [392] option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)

- Have you tried `mysqladmin -u root processlist` when `mysqld` appears to be running but not responding? Sometimes `mysqld` is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.

- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while you run your other queries.

- Try the following:

  1. Start `mysqld` from `gdb` (or another debugger). See Section 18.4, “Porting to Other Systems”.

  2. Run your test scripts.

  3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

     ```
     backtrace
     ```

```
info local
up
info local
up
info local
```

With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where `N` is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See Section 1.8, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.
- Do not rule out your server hardware when diagnosing problems. Defective hardware can be the cause of data corruption. Particular attention should be paid to your memory and disk subsystems when troubleshooting hardware.

B.5.4.3 How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and, as of MySQL 4.0.22, to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to `MyISAM` tables. As of MySQL 4.1.9, it also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition enables the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA INFILE` or after an `ALTER TABLE` statement. All of

these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

B.5.4.4 Where MySQL Stores Temporary Files

On Unix, MySQL uses the value of the `TMPDIR` environment variable as the path name of the directory in which to store temporary files. If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp`.

On Windows, Netware and OS2, MySQL checks in order the values of the `TMPDIR`, `TEMP`, and `TMP` environment variables. For the first one found to be set, MySQL uses it and does not check those remaining. If none of `TMPDIR`, `TEMP`, or `TMP` are set, MySQL uses the Windows system default, which is usually `C:\windows\temp\`.

If the file system containing your temporary file directory is too small, you can use the `--tmpdir` [394] option to `mysqld` to specify a directory in a file system where you have enough space.

Starting from MySQL 4.1, the `--tmpdir` [394] option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (“`:`”) on Unix and semicolon characters (“`;`”) on Windows, NetWare, and OS/2.

Note

To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should not set `--tmpdir` [394] to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

MySQL arranges that temporary files are removed if `mysqld` is terminated. On platforms that support it (such as Unix), this is done by unlinking the file after opening it. The disadvantage of this is that the name does not appear in directory listings and you do not see a big temporary file that fills up the file system in which the temporary file directory is located. (In such cases, `lsof +L1` may be helpful in identifying large files associated with `mysqld`.)

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some `SELECT` queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form `SQL_*`.

`ALTER TABLE` creates a temporary table in the same directory as the original table.

B.5.4.5 How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On some versions of Unix, anyone can delete files in the `/tmp` directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your `/tmp` directory so that files can be deleted only by their owners or the superuser (`root`). To do this, set the `sticky` bit on the `/tmp` directory by logging in as `root` and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the `sticky` bit is set by executing `ls -ld /tmp`. If the last permission character is `t`, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in `/etc/my.cnf`:

  ```
  [mysqld]
  socket=/path/to/socket

  [client]
  socket=/path/to/socket
  ```

  See Section 4.2.3.3, “Using Option Files”.

- Specify a `--socket` [227] option on the command line to `mysqld_safe` and when you run client programs.

- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.

- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `--with-unix-socket-path` [96] option when you run `configure`. See Section 2.9.3, “MySQL Source-Configuration Options”.

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

B.5.4.6 Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` [841] returns the wrong value. This should be done for the environment in which the server runs; for example, in `mysqld_safe` or `mysql.server`. See Section 2.13, “Environment Variables”.

You can set the time zone for the server with the `--timezone=`*`timezone_name`* [245] option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The permissible values for `--timezone` [245] or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

B.5.5 Query-Related Issues

B.5.5.1 Case Sensitivity in String Searches

For nonbinary strings (`CHAR`, `VARCHAR`, `TEXT`), string searches use the collation of the comparison operands. For binary strings (`BINARY`, `VARBINARY`, `BLOB`), comparisons use the numeric values of the bytes in the operands; this means that for alphabetic characters, comparisons will be case sensitive.

A comparison between a nonbinary string and binary string is treated as a comparison of binary strings.

Simple comparison operations (`>=, >, =, <, <=`, sorting, and grouping) are based on each character's "sort value." Characters with the same sort value are treated as the same character. For example, if "e" and "é" have the same sort value in a given collation, they compare as equal.

The default character set and collation are `latin1` and `latin1_swedish_ci`, so nonbinary string comparisons are case insensitive by default. This means that if you search with *`col_name`* `LIKE 'a%'`, you get all column values that start with `A` or `a`. To make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See Section 12.1.5, "`CREATE TABLE` Syntax".

Before MySQL 4.1, `COLLATE` is unavailable. Use the `BINARY` [859] operator in expressions to treat a string as a binary string: `BINARY` *`col_name`* `LIKE 'a%'` or *`col_name`* `LIKE BINARY 'a%'`. In column declarations, use the `BINARY` attribute.

To cause a case-sensitive comparison of nonbinary strings to be case insensitive, use `COLLATE` to name a case-insensitive collation. The strings in the following example normally are case sensitive, but `COLLATE` changes the comparison to be case insensitive:

```
mysql> SET @s1 = 'MySQL' COLLATE latin1_bin,
    ->     @s2 = 'mysql' COLLATE latin1_bin;
mysql> SELECT @s1 = @s2;
+-----------+
| @s1 = @s2 |
+-----------+
|         0 |
+-----------+
mysql> SELECT @s1 COLLATE latin1_swedish_ci = @s2;
+-------------------------------------+
| @s1 COLLATE latin1_swedish_ci = @s2 |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

A binary string is case sensitive in comparisons. To compare the string as case insensitive, convert it to a nonbinary string and use `COLLATE` to name a case-insensitive collation:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
+--------------+
| @s = 'mysql' |
+--------------+
|            0 |
+--------------+
mysql> SELECT CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql';
+-------------------------------------------------------------------+
| CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql' |
+-------------------------------------------------------------------+
|                                                                 1 |
+-------------------------------------------------------------------+
```

To determine whether a value will compare as a nonbinary or binary string, use the `COLLATION()` [871] function. This example shows that `VERSION()` [876] returns a string that has a case-insensitive collation, so comparisons are case insensitive:

```
mysql> SELECT COLLATION(VERSION());
+----------------------+
| COLLATION(VERSION()) |
+----------------------+
| utf8_general_ci      |
+----------------------+
```

For binary strings, the collation value is `binary`, so comparisons will be case sensitive. One context in which you will see `binary` is for compression and encryption functions, which return binary strings as a general rule: string:

```
mysql> SELECT COLLATION(ENCRYPT('x')), COLLATION(SHA1('x'));
+-------------------------+----------------------+
| COLLATION(ENCRYPT('x')) | COLLATION(SHA1('x')) |
+-------------------------+----------------------+
| binary                  | binary               |
+-------------------------+----------------------+
```

B.5.5.2 Problems Using `DATE` Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is permitted. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context and vice versa. MySQL also permits a “relaxed” string format when updating and in a `WHERE` clause that compares a date to a `DATE`, `DATETIME`, or `TIMESTAMP` column. “Relaxed” format means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent. MySQL can also convert a string containing no separators (such as `'20040815'`), provided it makes sense as a date.

When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` to a constant string with the `<`, `<=`, `=`, `>=`, `>`, or `BETWEEN` operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns

- When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` column to an expression
- When you use any comparison method other than those just listed, such as `IN` or `STRCMP()` [807].

For those exceptions, the comparison is done by converting the objects to strings and performing a string comparison.

To be on the safe side, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special "zero" date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When a `'0000-00-00'` date is used through Connector/ODBC, it is automatically converted to `NULL` because ODBC cannot handle that kind of date.

Because MySQL performs the conversions just described, the following statements work (assume that `idate` is a `DATE` column):

```
INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');
INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';
```

However, the following statement does not work:

```
SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` [807] is a string function, so it converts `idate` to a string in `'YYYY-MM-DD'` format and performs a string comparison. It does not convert `'20030505'` to the date `'2003-05-05'` and perform a date comparison.

If a date to be stored in a `DATE` column cannot be converted to any reasonable value, MySQL stores `'0000-00-00'`.

B.5.5.3 Problems with `NULL` Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `''`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as "phone number is not known" and the meaning of the second can be regarded as "the person is known to have no phone, and thus no phone number."

To help with `NULL` handling, you can use the `IS NULL` [783] and `IS NOT NULL` [783] operators and the `IFNULL()` [791] function.

In SQL, the NULL value is never true in comparison to any other value, even NULL. An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return NULL:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

To search for column values that are NULL, you cannot use an expr = NULL test. The following statement returns no rows, because expr = NULL is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for NULL values, you must use the IS NULL [783] test. The following statements show how to find the NULL phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See Section 3.3.4.6, “Working with NULL Values”, for additional information and examples.

You can add an index on a column that can have NULL values if you are using MySQL 3.23.2 or newer and are using the MyISAM, InnoDB, or BDB storage engine. As of MySQL 4.0.2, the MEMORY storage engine also supports NULL values in indexes. Otherwise, you must declare an indexed column NOT NULL and you cannot insert NULL into the column.

When reading data with LOAD DATA INFILE, empty or missing columns are updated with ''. To load a NULL value into a column, use \N in the data file. The literal word “NULL” may also be used under some circumstances. See Section 12.2.5, “LOAD DATA INFILE Syntax”.

When using DISTINCT, GROUP BY, or ORDER BY, all NULL values are regarded as equal.

When using ORDER BY, NULL values are presented first, or last if you specify DESC to sort in descending order. Exception: In MySQL 4.0.2 through 4.0.10, NULL values sort first regardless of sort order.

Aggregate (summary) functions such as COUNT() [882], MIN() [884], and SUM() [884] ignore NULL values. The exception to this is COUNT(*) [882], which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-NULL values in the age column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles NULL values specially. If you insert NULL into a TIMESTAMP column, the current date and time is inserted. If you insert NULL into an integer or floating-point column that has the AUTO_INCREMENT attribute, the next number in the sequence is inserted.

B.5.5.4 Problems with Column Aliases

An alias can be used in a query select list to give a column a different name. You can use the alias in GROUP BY, ORDER BY, or HAVING clauses to refer to the column:

```
SELECT SQRT(a*b) AS root FROM tbl_name
  GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
  GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL disallows references to column aliases in a `WHERE` clause. This restriction is imposed because when the `WHERE` clause is evaluated, the column value may not yet have been determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
  WHERE cnt > 0 GROUP BY id;
```

The `WHERE` clause determines which rows should be included in the `GROUP BY` clause, but it refers to the alias of a column value that is not known until after the rows have been selected, and grouped by the `GROUP BY`.

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
SELECT 1 AS `one`, 2 AS 'two';
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal. For example, this statement groups by the values in column `id`, referenced using the alias `` `a` ``:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY `a`;
```

But this statement groups by the literal string `'a'` and will not work as expected:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY 'a';
```

B.5.5.5 Rollback Failure for Nontransactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These nontransactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and nontransactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` does not support a storage engine, it instead creates the table as a `MyISAM` table, which is nontransactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See Section 12.4.5.23, “`SHOW TABLE STATUS` Syntax”, and Section 12.4.5.7, “`SHOW CREATE TABLE` Syntax”.

You can check which storage engines your `mysqld` server supports by using this statement:

```
```

```
SHOW ENGINES;
```

Before MySQL 4.1.2, `SHOW ENGINES` is unavailable. Use the following statement instead and check the value of the variable that is associated with the storage engine in which you are interested:

```
SHOW VARIABLES LIKE 'have_%';
```

For example, to determine whether the `InnoDB` storage engine is available, check the value of the `have_innodb` [413] variable.

See Section 12.4.5.10, "`SHOW ENGINES` Syntax", and Section 12.4.5.25, "`SHOW VARIABLES` Syntax".

B.5.5.6 Deleting Rows from Related Tables

MySQL does not support subqueries prior to version 4.1, or the use of more than one table in the `DELETE` statement prior to version 4.0. If your version of MySQL does not support subqueries or multiple-table `DELETE` statements, you can use the following approach to delete rows from two related tables:

1. `SELECT` the rows based on some `WHERE` condition in the main table.
2. `DELETE` the rows in the main table based on the same condition.
3. `DELETE FROM related_table WHERE related_column IN (selected_rows)`.

If the total length of the `DELETE` statement for `related_table` is more than 1MB (the default value of the `max_allowed_packet` [418] system variable), you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

B.5.5.7 Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that returns no rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See Section 12.7.2, "`EXPLAIN` Syntax".
2. Select only those columns that are used in the `WHERE` clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are large, it is a good idea to use `LIMIT 10` with the query.
4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.
5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you cannot use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the `FLOAT` to a `DOUBLE` fixes this. See Section B.5.5.8, "Problems with Floating-Point Values".

   Similar problems may be encountered when comparing `DECIMAL` values.
6. If you still can't figure out what is wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in

an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

Verify that the test file demonstrates the problem by executing these commands:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Attach the test file to a bug report, which you can file using the instructions in Section 1.8, "How to Report Bugs or Problems".

B.5.5.8 Problems with Floating-Point Values

Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The `FLOAT` and `DOUBLE` data types are subject to these issues. `DECIMAL` columns store values with exact precision because they are represented as strings, but calculations on `DECIMAL` values are done using floating-point operations.

The following example demonstrates the problem. It shows that even for older `DECIMAL` columns, calculations that are done using floating-point operations are subject to floating-point error. (Were you to replace the `DECIMAL` columns with `FLOAT`, similar problems would occur.)

```
mysql> CREATE TABLE t1 (i INT, d1 DECIMAL(9,2), d2 DECIMAL(9,2));
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
    -> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
    -> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
    -> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
    -> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
    -> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+-------+
| i    | a      | b     |
+------+--------+-------+
|    1 |  21.40 | 21.40 |
|    2 |  76.80 | 76.80 |
|    3 |   7.40 |  7.40 |
|    4 |  15.40 | 15.40 |
|    5 |   7.20 |  7.20 |
|    6 | -51.40 |  0.00 |
+------+--------+-------+
```

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

As of MySQL 5.0.3, you will get only the last row in the above result.

The problem cannot be solved by using `ROUND()` [823] or similar functions, because the result is still a floating-point number:

```
mysql> SELECT i, ROUND(SUM(d1), 2) AS a, ROUND(SUM(d2), 2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
```

```
+------+--------+-------+
| i    | a      | b     |
+------+--------+-------+
|    1 |  21.40 | 21.40 |
|    2 |  76.80 | 76.80 |
|    3 |   7.40 |  7.40 |
|    4 |  15.40 | 15.40 |
|    5 |   7.20 |  7.20 |
|    6 | -51.40 |  0.00 |
+------+--------+-------+
```

This is what the numbers in column a look like when displayed with more decimal places:

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1.0000000000000000 AS a,
    -> ROUND(SUM(d2), 2) AS b FROM t1 GROUP BY i HAVING a <> b;
+------+----------------------+-------+
| i    | a                    | b     |
+------+----------------------+-------+
|    1 |  21.3999999999999986 | 21.40 |
|    2 |  76.7999999999999972 | 76.80 |
|    3 |   7.4000000000000004 |  7.40 |
|    4 |  15.4000000000000004 | 15.40 |
|    5 |   7.2000000000000002 |  7.20 |
|    6 | -51.3999999999999986 |  0.00 |
+------+----------------------+-------+
```

Depending on your computer architecture, you may or may not see similar results. For example, on some machines you may get the “correct” results by multiplying both arguments by 1, as the following example shows.

Warning

Never use this method in your applications. It is not an example of a trustworthy method!

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1 AS a, ROUND(SUM(d2), 2)*1 AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+------+
| i    | a      | b    |
+------+--------+------+
|    6 | -51.40 | 0.00 |
+------+--------+------+
```

The reason that the preceding example seems to work is that on the particular machine where the test was done, CPU floating-point arithmetic happens to round the numbers to the same value. However, there is no rule that any CPU should do so, so this method cannot be trusted.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) > 0.0001;
+------+--------+------+
| i    | a      | b    |
+------+--------+------+
|    6 | -51.40 | 0.00 |
+------+--------+------+
```

```
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+------+-------+-------+
| i    | a     | b     |
+------+-------+-------+
|    1 | 21.40 | 21.40 |
|    2 | 76.80 | 76.80 |
|    3 |  7.40 |  7.40 |
|    4 | 15.40 | 15.40 |
|    5 |  7.20 |  7.20 |
+------+-------+-------+
```

Floating-point values are subject to platform or implementation dependencies. Suppose that you execute the following statements:

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52','-1e+52');
SELECT * FROM t1;
```

On some platforms, the `SELECT` statement returns `inf` and `-inf`. On others, it returns `0` and `-0`.

An implication of the preceding issues is that if you attempt to create a replication slave by dumping table contents with `mysqldump` on the master and reloading the dump file into the slave, tables containing floating-point columns might differ between the two hosts.

B.5.6 Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL does not have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  `EXPLAIN` is discussed in more detail in Section 12.7.2, “`EXPLAIN` Syntax”.

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 12.4.2.1, “`ANALYZE TABLE` Syntax”.

- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
  ```

  `USE INDEX` and `IGNORE INDEX` may also be useful. See Section 12.2.7.2, “Index Hint Syntax”.

- Global and table-level `STRAIGHT_JOIN`. See Section 12.2.7, “`SELECT` Syntax”.

- You can tune global or thread-specific system variables. For example, start `mysqld` with the `--max-seeks-for-key=1000` [420] option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.3, “Server System Variables”.

B.5.7 Table Definition-Related Issues

B.5.7.1 Problems with `ALTER TABLE`

Before MySQL 4.1, `ALTER TABLE` changes a table to the current character set. As of 4.1, `ALTER TABLE` changes character sets only if you request it explicitly. If you get a duplicate-key error when `ALTER TABLE` changes the character set or collation of a character column, the cause is either that the new column collation maps two keys to the same value or that the table is corrupted. In the latter case, you should run `REPAIR TABLE` on the table.

If `ALTER TABLE` dies with the following error, the problem may be that MySQL crashed during an earlier `ALTER TABLE` operation and there is an old table named `A-xxx` or `B-xxx` lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with `A-` or `B-`. (You may want to move them elsewhere instead of deleting them.)

`ALTER TABLE` works in the following way:

- Create a new table named `A-xxx` with the requested structural changes.
- Copy all rows from the original table to `A-xxx`.
- Rename the original table to `B-xxx`.
- Rename `A-xxx` to your original table name.
- Delete `B-xxx`.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as `B-xxx`. A simple rename of the table files at the system level should get your data back.

If you use `ALTER TABLE` on a transactional table or if you are using Windows or OS/2, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

B.5.7.2 `TEMPORARY` Table Problems

The following list indicates limitations on the use of `TEMPORARY` tables:

- A `TEMPORARY` table can only be of type `HEAP`, `ISAM`, `MyISAM`, `MERGE`, or `InnoDB`.

  Temporary tables are not supported for MySQL Cluster.

- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The `SHOW TABLES` statement does not list `TEMPORARY` tables.
- You cannot use `RENAME` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

- There are known issues in using temporary tables with replication. See Section 14.7, "Replication Features and Issues", for more information.

B.5.8 Known Issues in MySQL

This section lists known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in Section 2.12, "Operating System-Specific Notes", and Section 18.4, "Porting to Other Systems".

B.5.8.1 Issues in MySQL 3.23 Fixed in a Later MySQL Version

The following known issues have not been fixed in MySQL 3.23 for various reasons, and are not classified as critical.

- Fixed in MySQL 4.0: Avoid using spaces at the end of column names because this can cause unexpected behavior. (Bug #4196)
- Fixed in MySQL 4.0.12: You can get a deadlock (hung thread) if you use `LOCK TABLE` to lock multiple tables and then in the same connection use `DROP TABLE` to drop one of them while another thread is trying to lock it. (To break the deadlock, you can use `KILL` to terminate any of the threads involved.)
- Fixed in MySQL 4.0.11: `SELECT MAX(key_column) FROM t1,t2,t3...` where one of the tables are empty doesn't return `NULL` but instead returns the maximum value for the column.
- `DELETE FROM heap_table` without a `WHERE` clause doesn't work on a locked `HEAP` table.

B.5.8.2 Issues in MySQL 4.0 Fixed in a Later Version

The following known issues have not been fixed in MySQL 4.0 for various reasons, and are not classified as critical.

- Fixed in MySQL 4.1.10: Using `HAVING`, you can get a crash or wrong result if you use an alias to a `RAND()` [822] function. This will not be fixed in 4.0 because the fix may break compatability with some applications.
- Fixed in MySQL 4.1.1: In a `UNION`, the first `SELECT` determines the type, `max_length`, and `NULL` properties for the resulting columns.
- Fixed in MySQL 4.1: In `DELETE` with many tables, you can't refer to tables to be deleted through an alias.
- Fixed in MySQL 4.1.2: You cannot mix `UNION ALL` and `UNION DISTINCT` in the same query. If you use `ALL` for one `UNION`, it is used for all of them.
- `FLUSH TABLES WITH READ LOCK` does not block `CREATE TABLE`, which may cause a problem with the binary log position when doing a full backup of tables and the binary log.
- Fixed in MySQL 4.1.8: `mysqldump --single-transaction --master-data` behaved like `mysqldump --master-data`, so the dump was a blocking one.

- When using the `RPAD()` [800] function (or any function adding spaces to the right) in a query that had to be resolved by using a temporary table, all resulting strings had rightmost spaces removed (that is, `RPAD()` [800] did not work).

B.5.8.3 Issues in MySQL 4.1 Fixed in a Later Version

The following known issues have not been fixed in MySQL 4.1 for various reasons, and are not classified as critical.

- Fixed in 5.0.3: `VARCHAR` and `VARBINARY` did not remember end space.

B.5.8.4 Open Issues in MySQL

The following problems are known:

- If you compare a `NULL` value to a subquery using `ALL`, `ANY`, or `SOME`, and the subquery returns an empty result, the comparison might evaluate to the nonstandard result of `NULL` rather than to `TRUE` or `FALSE`. This issue has been fixed in MySQL 5.0 and later (Bug #8804).
- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` [872] or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint doesn't work in replication because the constraint may have another name on the slave.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` doesn't work inside `GROUP_CONCAT()` [883] if you don't use all and only those columns that are in the `DISTINCT` list.
- If one user has a long-running transaction and another user drops a table that is updated in the transaction, there is small chance that the binary log may contain the `DROP TABLE` statement before the table is used in the transaction itself. We plan to fix this by having the `DROP TABLE` statement wait until the table is not being used in any transaction.
- When inserting a big integer value (between 2^{63} and 2^{64}–1) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- `FLUSH TABLES WITH READ LOCK` does not block `COMMIT` if the server is running without binary logging, which may cause a problem (of consistency between tables) when doing a full backup.
- `ANALYZE TABLE` on a `BDB` table may in some cases make the table unusable until you restart `mysqld`. If this happens, look for errors of the following form in the MySQL error file:

```
001207 22:07:56  bdb:  log_flush: LSN past current end-of-log
```

- Don't execute `ALTER TABLE` on a `BDB` table on which you are running multiple-statement transactions until all those transactions complete. (The transaction might be ignored.)
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` may cause problems on tables for which you are using `INSERT DELAYED`.
- Performing `LOCK TABLE ...` and `FLUSH TABLES ...` doesn't guarantee that there isn't a half-finished transaction in progress on the table.

- `BDB` tables are relatively slow to open. If you have many `BDB` tables in a database, it takes a long time to use the `mysql` client on the database if you are not using the `-A` option or if you are using `rehash`. This is especially noticeable when you have a large table cache.

- Replication uses query-level logging: The master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases.

  It is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is nondeterministic (generally not a recommended practice, even outside of replication).

  For example:

  - `CREATE TABLE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.

  - `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.

  - `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

  If and only if the preceding queries have no `ORDER BY` clause guaranteeing a deterministic order.

  For example, for `INSERT ... SELECT` with no `ORDER BY`, the `SELECT` may return rows in a different order (which results in a row having different ranks, hence getting a different number in the `AUTO_INCREMENT` column), depending on the choices made by the optimizers on the master and slave.

  A query is optimized differently on the master and slave only if:

  - The files used by the two queries are not exactly the same; for example, `OPTIMIZE TABLE` was run on the master tables and not on the slave tables. (To fix this, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE` are written to the binary log as of MySQL 4.1.1).

  - The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use `InnoDB` on the master, but `MyISAM` on the slave if the slave has less available disk space.)

  - MySQL buffer sizes (`key_buffer_size` [415], and so on) are different on the master and slave.

  - The master and slave run different MySQL versions, and the optimizer code differs between these versions.

  This problem may also affect database restoration using `mysqlbinlog|mysql`.

  The easiest way to avoid this problem is to add an `ORDER BY` clause to the aforementioned nondeterministic queries to ensure that the rows are always stored or modified in the same order.

  In future MySQL versions, we will automatically add an `ORDER BY` clause when needed.

The following issues are known and will be fixed in due time:

- Log file names are based on the server host name (if you don't specify a file name with the startup option). You have to use options such as `--log-bin=`*`old_host_name`*`-bin` [1181] if you change your host name to something else. Another option is to rename the old files to reflect your host name change (if these are binary logs, you need to edit the binary log index file and fix the binary log file names there as well). See Section 5.1.2, “Server Command Options”.

- `mysqlbinlog` does not delete temporary files left after a `LOAD DATA INFILE` statement. See Section 4.6.6, “`mysqlbinlog` — Utility for Processing Binary Log Files”.
- `RENAME` doesn't work with `TEMPORARY` tables or tables used in a `MERGE` table.
- Due to the way table format (`.frm`) files are stored, you cannot use character 255 (`CHAR(255)`) in table names, column names, or enumerations. This is scheduled to be fixed in version 5.1 when we implement new table definition format files.
- When using `SET CHARACTER SET`, you can't use translated characters in database, table, and column names.
- You can't use “_” or “%” with `ESCAPE` in `LIKE ... ESCAPE` [804].
- If you have a `DECIMAL` column in which the same number is stored in different formats (for example, `+01.00`, `1.00`, `01.00`), `GROUP BY` may regard each value as a different value.
- You cannot build the server in another directory when using MIT-pthreads. Because this requires changes to MIT-pthreads, we are not likely to fix this. See Section 2.9.6, “MIT-pthreads Notes”.
- `BLOB` and `TEXT` values can't “reliably” be used in `GROUP BY`, `ORDER BY` or `DISTINCT`. Only the first `max_sort_length` [420] bytes are used when comparing `BLOB` values in these cases. The default value of `max_sort_length` [420] value is 1024 and can be changed at server startup time. As of MySQL 4.0.3, it can be changed at runtime. For older versions, a workaround is to use a substring. For example:

```
SELECT DISTINCT LEFT(blob_col,2048) FROM tbl_name;
```

- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF()` [790] and `ELT()` [795] with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE` precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields. MySQL Server 4.0 has better `BIGINT` handling than 3.23.
- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()` [884], `MAX()` [884], and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.
- `mysqld_safe` redirects all messages from `mysqld` to the `mysqld` log. One problem with this is that if you execute `mysqladmin refresh` to close and reopen the log, `stdout` and `stderr` are still redirected to the old log. If you use the general query log extensively, you should edit `mysqld_safe` to log to *`host_name`*`.err` instead of *`host_name`*`.log` so that you can easily reclaim the space for the old log by deleting it and executing `mysqladmin refresh`.
- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by `2`, **not** `1`:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following doesn't work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns don't participate in the `DISTINCT` comparison. We will probably change this in the future to never compare the hidden columns when executing `DISTINCT`.

  An example of this is:

  ```
  SELECT DISTINCT mp3id FROM band_downloads
         WHERE userid = 9 ORDER BY id DESC;
  ```

  and

  ```
  SELECT DISTINCT band_downloads.mp3id
         FROM band_downloads,band_mp3
         WHERE band_downloads.userid = 9
         AND band_mp3.id = band_downloads.mp3id
         ORDER BY band_downloads.id DESC;
  ```

  In the second case, using MySQL Server 3.23.x, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

  Note that this happens only for queries where that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.

- Creation of a table of type `MERGE` doesn't check whether the underlying tables are compatible types.

- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

Appendix C MySQL Release Notes

Table of Contents

This appendix lists the changes from version to version in the MySQL source code through the latest version of MySQL 4.1.

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a recent version of MySQL listed here that you can't find on our download page (http://dev.mysql.com/downloads/), it means that the version has not yet been released.

The date mentioned with a release version is the date of the last Bazaar commit on which the release was based, not the date when the packages were made available. The binaries are usually made available a few days after the date of the tagged ChangeSet, because building and testing all packages takes some time.

The manual included in the source and binary distributions may not be fully accurate when it comes to the release changelog entries, because the integration of the manual happens at build time. For the most up-to-date release changelog, please refer to the online version instead.

C.1 Changes in Release 4.1.x (Lifecycle Support Ended)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

The following list summarizes the features in MySQL Server 4.1 that are not present in previous versions. For a full list of changes, please refer to the changelog sections for individual 4.1 releases.

- The `SUBSTRING()` [802] function can now take a negative value for the *`pos`* (position) argument. See Section 11.5, “String Functions”.

- Subqueries and derived tables (unnamed views). See Section 12.2.8, “Subquery Syntax”.
- `INSERT ... ON DUPLICATE KEY UPDATE ...` syntax. This enables you to `UPDATE` an existing row if the insert would cause a duplicate value in a `PRIMARY` or `UNIQUE` key. (`REPLACE` enables you to overwrite an existing row, which is something entirely different.) See Section 12.2.4, “`INSERT` Syntax”.
- A newly designed `GROUP_CONCAT()` [883] aggregate function. See Section 11.15, “Functions and Modifiers for Use with `GROUP BY` Clauses”.
- Extensive Unicode (UTF8) support.
- Table names and column names now are stored in `UTF8`. This makes MySQL more flexible, but might cause some problems upgrading if you have table or column names that use characters outside of the standard 7-bit US-ASCII range. See Section 2.11.1.1, “Upgrading from MySQL 4.0 to 4.1”.
- Character sets can be defined per column, table, and database.
- New key cache for `MyISAM` tables with many tunable parameters. You can have multiple key caches, preload index into caches for batches...
- `BTREE` index on `HEAP` tables.
- Support for OpenGIS spatial types (geographical data). See Chapter 16, *Spatial Extensions*.
- `SHOW WARNINGS` shows warnings for the last command. See Section 12.4.5.26, “`SHOW WARNINGS` Syntax”.
- Faster binary protocol with prepared statements and parameter binding. See Section 17.6.7, “C API Prepared Statements”.
- You can now issue multiple statements with a single C API call and then read the results in one go. See Section 17.6.15, “C API Support for Multiple Statement Execution”.
- Create Table: `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table2 LIKE table1`.
- Server based `HELP` statement that can be used in the `mysql` command-line client (and other clients) to get help for SQL statements.

C.1.1 Changes in MySQL 4.1.25 (2008-12-01)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.1 release family.

Functionality Added or Changed

- **Security Enhancement:** To enable stricter control over the location from which user-defined functions can be loaded, the `plugin_dir` [423] system variable has been backported from MySQL 5.1. If the value is nonempty, user-defined function object files can be loaded only from the directory named by this variable. If the value is empty, the behavior that is used prior to the inclusion of `plugin_dir` [423] applies: The UDF object files must be located in a directory that is searched by your system's dynamic linker.

  If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making

`plugin_dir` [423] read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely. (Bug #37428)

Bugs Fixed

- **Security Fix; Important Change:** Additional corrections were made for the symlink-related privilege problem originally addressed in MySQL 4.1.24. The original fix did not correctly handle the data directory path name if it contained symlinked directories in its path, and the check was made only at table-creation time, not at table-opening time later. (Bug #32167, CVE-2008-2079)

  References: See also Bug #39277.

- On Windows, the installer attempted to use JScript to determine whether the target data directory already existed. On Windows Vista x64, this resulted in an error because the installer was attempting to run the JScript in a 32-bit engine, which wasn't registered on Vista. The installer no longer uses JScript but instead relies on a native WiX command. (Bug #36103)

- The Windows installer displayed incorrect product names in some images. (Bug #40845)

- `INSERT INTO ... SELECT` caused a crash if `innodb_locks_unsafe_for_binlog` [1070] was enabled. (Bug #27294)

- The MySQL Instance Configuration Wizard would not permit you to choose a service name, even though the criteria for the service name were valid. The code that checks the name has been updated to support the correct criteria of any string less than 256 character and not containing either a forward or backward slash character. (Bug #27013)

C.1.2 Changes in MySQL 4.1.24 (2008-03-01)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.1 release family.

Functionality Added or Changed

- The `ndbd` and `ndb_mgmd` man pages have been reclassified from volume 1 to volume 8. (Bug #34642)

Bugs Fixed

- **Security Fix; Important Change:** It was possible to circumvent privileges through the creation of `MyISAM` tables employing the `DATA DIRECTORY` and `INDEX DIRECTORY` options to overwrite existing table files in the MySQL data directory. Use of the MySQL data directory in `DATA DIRECTORY` and `INDEX DIRECTORY` path name is no longer permitted.

  Note

  Additional fixes were made in MySQL 4.1.25.

  (Bug #32167, CVE-2008-2079)

  References: See also Bug #39277.

- **Security Fix:** Using `RENAME TABLE` against a table with explicit `DATA DIRECTORY` and `INDEX DIRECTORY` options can be used to overwrite system table information by replacing the symbolic link points. the file to which the symlink points.

MySQL will now return an error when the file to which the symlink points already exists. (Bug #32111, CVE-2007-5969)

- **Security Fix:** A malformed password packet in the connection protocol could cause the server to crash. Thanks for Dormando for reporting this bug, and for providing details and a proof of concept. (Bug #28984, CVE-2007-3780)
- **Security Enhancement:** It was possible to force an error message of excessive length which could lead to a buffer overflow. This has been made no longer possible as a security precaution. (Bug #32707)
- **Replication:** Connections from one `mysqld` server to another failed on Mac OS X, affecting replication and `FEDERATED` tables. (Bug #29083)

  References: See also Bug #26664.
- An internal buffer in `mysql` was too short. Overextending it could cause stack problems or segmentation violations on some architectures. (This is not a problem that could be exploited to run arbitrary code.) (Bug #33841)
- `make_binary_distribution` passed the `--print-libgcc-file` option to the C compiler, but this does not work with the `ICC` compiler. (Bug #33536)
- Performing a full text search on a table could cause a crash on a 64-bit platforms with certain characteristics. Searches that were known to cause a crash with certain datasets included numeric values and strings where the match string included data enclosed in single or double quotation marks. (Bug #11392)
- `PURGE MASTER LOGS BEFORE (subquery)` caused a server crash. Subqueries are forbidden in the `BEFORE` clause now. (Bug #28553)
- `mysql_setpermission` tried to grant global-only privileges at the database level. (Bug #14618)
- A field packet with `NULL` fields caused a `libmysqlclient` crash. (Bug #29494)
- On 64-bit Windows systems, the Config Wizard failed to complete the setup because 64-bit Windows does not resolve dynamic linking of the 64-bit `libmysql.dll` to a 32-bit application like the Config Wizard. (Bug #14649)
- When one thread attempts to lock two (or more) tables and another thread executes a statement that aborts these locks (such as `REPAIR TABLE`, `OPTIMIZE TABLE`, or `CHECK TABLE`), the thread might get a table object with an incorrect lock type in the table cache. The result is table corruption or a server crash. (Bug #28574)
- Issuing a `DELETE` statement having both an `ORDER BY` clause and a `LIMIT` clause could cause `mysqld` to crash. (Bug #30385)
- If an `ENUM` column contained `''` as one of its members (represented with numeric value greater than 0), and the column contained error values (represented as 0 and displayed as `''`), using `ALTER TABLE` to modify the column definition caused the 0 values to be given the numeric value of the nonzero `''` member. (Bug #29251)
- The semantics of `BIGINT` depended on platform-specific characteristics. (Bug #29079)
- Using up-arrow for command-line recall in `mysql` could cause a segmentation fault. (Bug #10218)
- Dropping a user-defined function could cause a server crash if the function was still in use by another thread. (Bug #27564)

- Adding `DISTINCT` could cause incorrect rows to appear in a query result. (Bug #29911)
- In some cases, `INSERT INTO ... SELECT ... GROUP BY` could insert rows even if the `SELECT` by itself produced an empty result. (Bug #29717)
- Error returns from the `time()` system call were ignored. (Bug #27198)
- If one thread was performing concurrent inserts, other threads reading from the same table using equality key searches could see the index values for new rows before the data values had been written, leading to reports of table corruption. (Bug #29838)
- A network structure was initialized incorrectly, leading to embedded server crashes. (Bug #29117)
- `SELECT ... INTO OUTFILE` followed by `LOAD DATA` could result in garbled characters when the `FIELDS ENCLOSED BY` clause named a delimiter of `'0'`, `'b'`, `'n'`, `'r'`, `'t'`, `'N'`, or `'Z'` due to an interaction of character encoding and doubling for data values containing the enclosed-by character. (Bug #29294)
- Format strings in English error messages were insufficiently wide for path names printed in those messages by the embedded server. (Bug #16635)
- On Mac OS X, shared-library installation path names were incorrect. (Bug #28544)
- For `MEMORY` tables, `DELETE` statements that remove rows based on an index read could fail to remove all matching rows. (Bug #30590)
- For `InnoDB` tables that use the `utf8` character set, incorrect results could occur for DML statements such as `DELETE` or `UPDATE` that use an index on character-based columns. (Bug #28878)

  References: See also Bug #29449, Bug #30485, Bug #31395. This bug was introduced by Bug #13195.
- With small values of `myisam_sort_buffer_size` [421], `REPAIR TABLE` for `MyISAM` tables could cause a server crash. (Bug #31174)
- Internal conversion routines could fail for several multi-byte character sets (`big5`, `cp932`, `euckr`, `gb2312`, `sjis`) for empty strings or during evaluation of `SOUNDS LIKE` [801]. (Bug #31069, Bug #31070)
- Versions of `mysqldump` from MySQL 4.1 or higher tried to use `START TRANSACTION WITH CONSISTENT SNAPSHOT` if the `--single-transaction` [298] and `--master-data` [296] options were given, even with servers older than 4.1 that do not support consistent snapshots. (Bug #30444)
- A buffer used when setting variables was not dimensioned to accommodate the trailing `'\0'` byte, so a single-byte buffer overrun was possible. (Bug #31588)
- Setting certain values on a table using a spatial index could cause the server to crash. (Bug #30286)
- The `GeomFromText()` [1397] function could cause a server crash if the first argument was `NULL` or the empty string. (Bug #30955)
- The server crashed on optimizations involving a join of `INT` and `MEDIUMINT` columns and a system variable in the `WHERE` clause. (Bug #32103)
- Full-text searches on `ucs2` columns caused a server crash. (`FULLTEXT` indexes on `ucs2` columns cannot be used, but it should be possible to perform `IN BOOLEAN MODE` searches on `ucs2` columns without a crash.) (Bug #31159)
- For an almost-full `MyISAM` table, an insert that failed could leave the table in a corrupt state. (Bug #31305)

- The server could crash during `filesort` for `ORDER BY` based on expressions with `INET_NTOA()` [878] or `OCT()` [821] if those functions returned `NULL`. (Bug #31758)
- `myisamchk --unpack` could corrupt a table that when unpacked has static (fixed-length) row format. (Bug #31277)
- Data in `BLOB` or `GEOMETRY` columns could be cropped when performing a `UNION` query. (Bug #31158)
- Tables with a `GEOMETRY` column could be marked as corrupt if you added a non-`SPATIAL` index on a `GEOMETRY` column. (Bug #30284)
- On some 64-bit systems, inserting the largest negative value into a `BIGINT` column resulted in incorrect data. (Bug #30069)
- With `lower_case_table_names` [418] set, `CREATE TABLE LIKE` was treated differently by `libmysqld` than by the nonembedded server. (Bug #32063)
- `ucs2` does not work as a client character set, but attempts to use it as such were not rejected. Now `character_set_client` [408] cannot be set to `ucs2`. This also affects statements such as `SET NAMES` and `SET CHARACTER SET`. (Bug #31615)
- Denormalized double-precision numbers cannot be handled properly by old MIPS processors. For IRIX, this is now handled by enabling a mode to use a software workaround. (Bug #29085)
- The MySQL preferences pane did not work to start or stop MySQL on Mac OS X 10.5 (Leopard). (Bug #28854)
- On Mac OS X, the StartupItem for MySQL did not work. (Bug #25008)

C.1.3 Changes in MySQL 4.1.23 (2007-06-12)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.1 release family.

Functionality Added or Changed

- **Incompatible Change:** Previously, the `DATE_FORMAT()` [832] function returned a binary string. Now it returns a string with a character set and collation given by `character_set_connection` [408] and `collation_connection` [409] so that it can return month and weekday names containing non-ASCII characters. (Bug #22646)
- **Incompatible Change:** The `prepared_stmt_count` [423] system variable has been converted to the `Prepared_stmt_count` [453] global status variable (viewable with the `SHOW GLOBAL STATUS` statement). (Bug #23159)
- **Important Change:** When using a `MERGE` table, the definition of the table and the underlying `MyISAM` tables are checked each time the tables are opened for access (including any `SELECT` or `INSERT` statement). Each table is compared for column order, types, sizes, and associated indexes. If there is a difference in any one of the tables, the statement will fail.
- The `--memlock` [389] option relies on system calls that are unreliable on some operating systems. If a crash occurs, the server now checks whether `--memlock` [389] was specified and if so issues some information about possible workarounds. (Bug #22860)

- If the user specified the server options `--max-connections=N` [419] or `--table-cache=M` [431], a warning would be given in some cases that some values were recalculated, with the result that `--table-cache` [431] could be assigned greater value.

  In such cases, both the warning and the increase in the `--table-cache` [431] value were completely harmless. Note also that it is not possible for the MySQL Server to predict or to control limitations on the maximum number of open files, since this is determined by the operating system.

  The value of `--table-cache` [431] is no longer increased automatically, and a warning is now given only if some values had to be decreased due to operating system limits. (Bug #21915)

- The server now includes a timestamp in error messages that are logged as a result of unhandled signals (such as `mysqld got signal 11` messages). (Bug #24878)

- `mysqldump --single-transaction` now uses `START TRANSACTION /*!40100 WITH CONSISTENT SNAPSHOT */` rather than `BEGIN` to start a transaction, so that a consistent snapshot will be used on those servers that support it. (Bug #19660)

- `INSERT DELAYED` statements on `BLACKHOLE` tables are now rejected, due to the fact that the `BLACKHOLE` storage engine does not support them. (Bug #27998)

- A dependency on the Intel runtime libraries existed in the `shared-xxx` RPMs for the IA-64 CPU of some versions of MySQL 4.1 (4.1.16, 4.1.20, and 4.1.22). This has been resolved. (Bug #18776)

Bugs Fixed

- **Security Fix:** The requirement of the `DROP` [491] privilege for `RENAME TABLE` was not enforced. (Bug #27515, CVE-2007-2691)

- **Performance:** `InnoDB` showed substandard performance with multiple queries running concurrently. (Bug #15815)

- **Performance:** `InnoDB` exhibited thread thrashing with more than 50 concurrent connections under an update-intensive workload. (Bug #22868)

- **Incompatible Change:** `INSERT DELAYED` statements are not supported for `MERGE` tables, but the `MERGE` storage engine was not rejecting such statements, resulting in table corruption. Applications previously using `INSERT DELAYED` into `MERGE` table will break when upgrading to versions with this fix. To avoid the problem, remove `DELAYED` from such statements. (Bug #26464)

- **Incompatible Change:** For `ENUM` columns that had enumeration values containing commas, the commas were mapped to `0xff` internally. However, this rendered the commas indistinguishable from true `0xff` characters in the values. This no longer occurs. However, the fix requires that you dump and reload any tables that have `ENUM` columns containing any true `0xff` values. Dump the tables using `mysqldump` with the current server before upgrading from a version of MySQL 4.1 older than 4.1.23 to version 4.1.23 or newer. (Bug #24660)

- **MySQL Cluster:** In some circumstances, shutting down the cluster could cause connected `mysqld` processes to crash. (Bug #25668)

- **MySQL Cluster:** The management client command *`node_id`* `STATUS` displayed the message `Node` *`node_id`*`: not connected` when *`node_id`* was not the node ID of a data node.

Note

The `ALL STATUS` command in the cluster management client still displays status information for data nodes only. This is by design. See Section 15.5.2, “Commands in the MySQL Cluster Management Client”, for more information.

(Bug #21715)

- **MySQL Cluster:** When an API node sent more than 1024 signals in a single batch, `NDB` would process only the first 1024 of these, and then hang. (Bug #28443)
- **Replication:** Transient errors in replication from master to slave may trigger multiple `Got fatal error 1236: 'binlog truncated in the middle of event'` errors on the slave. (Bug #4053)
- **Replication:** Changes to the `lc_time_names` [417] system variable were not replicated. (Bug #22645)
- **Replication:** SQL statements close to the size of `max_allowed_packet` [418] could produce binary log events larger than `max_allowed_packet` [418] that could not be read by slave servers. (Bug #19402)
- **Replication:** `GRANT` statements were not replicated if the server was started with the `--replicate-ignore-table` [1177] or `--replicate-wild-ignore-table` [1178] option. (Bug #25482)
- **Replication:** If a slave server closed its relay log (for example, due to an error during log rotation), the I/O thread did not recognize this and still tried to write to the log, causing a server crash. (Bug #10798)
- **Cluster Replication:** Some queries that updated multiple tables were not backed up correctly. (Bug #27748)
- **Cluster API:** `libndbclient.so` was not versioned. (Bug #13522)
- When opening a corrupted `.frm` file during a query, the server crashes. (Bug #24358)
- `ISNULL(DATE(NULL))` [785] and `ISNULL(CAST(NULL AS DATE))` [785] erroneously returned false. (Bug #23938)
- The error message for error number `137` did not report which database/table combination reported the problem. (Bug #27173)
- A return value of `-1` from user-defined handlers was not handled well and could result in conflicts with server code. (Bug #24987)
- `X() IS NULL` and `Y() IS NULL` comparisons failed when `X()` [1404] and `Y()` [1404] returned `NULL`. (Bug #26038)
- `DOUBLE` values such as `20070202191048.000000` were being treated as illegal arguments by `WEEK()` [843]. (Bug #23616)
- The `mysqlserver.lib` library on Windows had many missing symbols. (Bug #29007)
- `LAST_DAY('0000-00-00')` [836] could cause a server crash. (Bug #23653)
- `SET lc_time_names = value` permitted only exact literal values, not expression values. (Bug #22647)
- The server could send incorrect column count information to the client for queries that produce a larger number of columns than can fit in a two-byte number. (Bug #19216)
- If there was insufficient memory to store or update a blob record in a `MyISAM` table then the table will marked as crashed. (Bug #23196)
- A server crash occurred when using `LOAD DATA` to load a table containing a `NOT NULL` spatial column, when the statement did not load the spatial column. Now a `NULL supplied to NOT NULL column` error occurs. (Bug #22372)

- If elements in a nontop-level `IN` subquery were accessed by an index and the subquery result set included a `NULL` value, the quantified predicate that contained the subquery was evaluated to `NULL` when it should return a non-`NULL` value. (Bug #23478)
- `mysql_fix_privilege_tables` did not accept a password containing embedded space or apostrophe characters. (Bug #17700)
- The `BUILD/check-cpu` script did not recognize Celeron processors. (Bug #20061)
- Accessing a fixed record format table with a crashed key definition results in server/`myisamchk` segmentation fault. (Bug #24855)
- If a thread previously serviced a connection that was killed, excessive memory and CPU use by the thread occurred if it later serviced a connection that had to wait for a table lock. (Bug #25966)
- The `MERGE` storage engine could return incorrect results when several index values that compare equality were present in an index (for example, `'gross'` and `'gross '`, which are considered equal but have different lengths). (Bug #24342)
- If `COMPRESS()` [866] returned `NULL`, subsequent invocations of `COMPRESS()` [866] within a result set or within a trigger also returned `NULL`. (Bug #23254)
- When updating a table that used a `JOIN` of the table itself (for example, when building trees) and the table was modified on one side of the expression, the table would either be reported as crashed or the wrong rows in the table would be updated. (Bug #21310)
- Referencing an ambiguous column alias in an expression in the `ORDER BY` clause of a query caused the server to crash. (Bug #25427)
- No warning was issued for use of the `DATA DIRECTORY` or `INDEX DIRECTORY` table options on a platform that does not support them. (Bug #17498)
- Duplicate entries were not assessed correctly in a `MEMORY` table with a `BTREE` primary key on a `utf8` `ENUM` column. (Bug #24985)
- `mysqldump --order-by-primary` failed if the primary key name was an identifier that required quoting. (Bug #13926)
- The internal functions for table preparation, creation, and alteration were not re-execution friendly, causing problems in code that: repeatedly altered a table; repeatedly created and dropped a table; opened and closed a cursor on a table, altered the table, and then reopened the cursor; used `ALTER TABLE` to change a table's current `AUTO_INCREMENT` value; created indexes on `utf8` columns.

  Re-execution of `CREATE DATABASE`, `CREATE TABLE`, and `ALTER TABLE` statements as prepared statements also caused incorrect results or crashes. (Bug #4968, Bug #6895, Bug #19182, Bug #19733, Bug #22060, Bug #24879)
- `SHOW COLUMNS` reported some `NOT NULL` columns as `NULL`. (Bug #22377)
- `STR_TO_DATE()` [838] returned `NULL` if the format string contained a space following a nonformat character. (Bug #22029)
- The arguments to the `ENCODE()` [867] and the `DECODE()` [866] functions were not printed correctly, causing problems in the output of `EXPLAIN EXTENDED` and in view definitions. (Bug #23409)
- Passing nested row expressions with different structures to an `IN` predicate caused a server crash. (Bug #27484)

- For `MyISAM` tables, `COUNT(*)` [882] could return an incorrect value if the `WHERE` clause compared an indexed `TEXT` column to the empty string (`''`). This happened if the column contained empty strings and also strings starting with control characters such as tab or newline. (Bug #26231)
- If an `ORDER BY` or `GROUP BY` list included a constant expression being optimized away and, at the same time, containing single-row subselects that returned more that one row, no error was reported. If a query required sorting by expressions containing single-row subselects that returned more than one row, execution of the query could cause a server crash. (Bug #24653)
- The return value from `my_seek()` was ignored. (Bug #22828)
- The second execution of a prepared statement from a `UNION` query with `ORDER BY RAND()` caused the server to crash. (Bug #27937)
- `LOAD DATA INFILE` sent an okay to the client before writing the binary log and committing the changes to the table had finished, thus violating ACID requirements. (Bug #26050)
- `NOW()` [837] returned the wrong value in statements executed at server startup with the `--init-file` [387] option. (Bug #23240)
- The fix for Bug #17212 provided correct sort order for misordered output of certain queries, but caused significant overall query performance degradation. (Results were correct (good), but returned much more slowly (bad).) The fix also affected performance of queries for which results were correct. The performance degradation has been addressed. (Bug #27531)
- `mysql_stmt_fetch()` did an invalid memory deallocation when used with the embedded server. (Bug #25492)
- For not-yet-authenticated connections, the `Time` column in `SHOW PROCESSLIST` was a random value rather than `NULL`. (Bug #23379)
- The `Handler_rollback` [452] status variable sometimes was incremented when no rollback had taken place. (Bug #22728)
- It was possible to use `DATETIME` values whose year, month, and day parts were all zeros but whose hour, minute, and second parts contained nonzero values, an example of such an illegal `DATETIME` being `'0000-00-00 11:23:45'`.

  Note

  This fix was reverted in MySQL 4.1.24.

  (Bug #21789)

  References: See also Bug #25301.
- The creation of MySQL system tables was not checked for by `mysql-test-run.pl`. (Bug #20166)
- For ODBC compatibility, MySQL supports use of `WHERE col_name IS NULL` for `DATE` or `DATETIME` columns that are `NOT NULL`, to permit column values of `'0000-00-00'` or `'0000-00-00 00:00:00'` to be selected. However, this was not working for `WHERE` clauses in `DELETE` statements. (Bug #23412)
- For `MERGE` tables defined on underlying tables that contained a short `VARCHAR` column (shorter than four characters), using `ALTER TABLE` on at least one but not all of the underlying tables caused the table definitions to be considered different from that of the `MERGE` table, even if the `ALTER TABLE` did not change the definition.

In addition, when the underlying tables contained a `TINYINT` or `CHAR(1)` column, the `MERGE` storage engine incorrectly reported that they differed from the `MERGE` table in certain cases. (Bug #26881)

- For `BOOLEAN` mode full-text searches on nonindexed columns, `NULL` rows generated by a `LEFT JOIN` caused incorrect query results. (Bug #14708, Bug #25637)
- Lack of validation for input and output `TIME` values resulted in several problems: `SEC_TO_TIME()` [838] in some cases did not clip large values to the `TIME` range appropriately; `SEC_TO_TIME()` [838] treated `BIGINT UNSIGNED` values as signed; only truncation warnings were produced when both truncation and out-of-range `TIME` values occurred. (Bug #11655, Bug #20927)
- Using `CAST()` [859] to convert `DATETIME` values to numeric values did not work. (Bug #23656)
- A reference to a nonexistent column in the `ORDER BY` clause of an `UPDATE ... ORDER BY` statement could cause a server crash. (Bug #25126)
- Selecting into variables sometimes returned incorrect wrong results. (Bug #20836)
- A deadlock could occur, with the server hanging on `Closing tables`, with a sufficient number of concurrent `INSERT DELAYED`, `FLUSH TABLES`, and `ALTER TABLE` operations. (Bug #23312)
- Metadata for columns calculated from scalar subqueries was limited to integer, double, or string, even if the actual type of the column was different. (Bug #11032)
- The result set of a query that used `WITH ROLLUP` and `DISTINCT` could lack some rollup rows (rows with `NULL` values for grouping attributes) if the `GROUP BY` list contained constant expressions. (Bug #24856)
- A crash of the MySQL Server could occur when unpacking a `BLOB` column from a row in a corrupted MyISAM table. This could happen when trying to repair a table using either `REPAIR TABLE` or `myisamchk`; it could also happen when trying to access such a “broken” row using statements like `SELECT` if the table was not marked as crashed. (Bug #22053)
- Added support for `--debugger=dbx` for `mysql-test-run.pl` and added support for `--debugger=devenv`, `--debugger=DevEnv`, and `--debugger=`*`/path/to/`*`devenv`. (Bug #26792)
- There was a race condition in the `InnoDB` `fil_flush_file_spaces()` function. (Bug #24089)

  References: This bug was introduced by Bug #15653.
- `EXPLAIN` for a query on an empty table immediately after its creation could result in a server crash. (Bug #28272)
- MySQL failed to build on Linux/Alpha. (Bug #23256)

  References: This bug was introduced by Bug #21250.
- Running `CHECK TABLE` concurrently with a `SELECT`, `INSERT` or other statement on Windows could corrupt a MyISAM table. (Bug #25712)
- Some small double precision numbers (such as `1.00000001e-300`) that should have been accepted were truncated to zero. (Bug #22129)
- `mysqld_multi` and `mysqlaccess` looked for option files in `/etc` even if the `--sysconfdir` option for `configure` had been given to specify a different directory. (Bug #24780)
- A compressed `MyISAM` table that became corrupted could crash `myisamchk` and possibly the MySQL Server. (Bug #23139)

- The `--extern` option for `mysql-test-run.pl` did not function correctly. (Bug #24354)
- `mysql-test-run` did not work correctly for RPM-based installations. (Bug #17194)
- `mysqltest` incorrectly tried to retrieve result sets for some queries where no result set was available. (Bug #19410)
- If there was insufficient memory available to `mysqld`, this could sometimes cause the server to hang during startup. (Bug #24751)
- Trailing spaces were not removed from Unicode `CHAR` column values when used in indexes. This resulted in excessive usage of storage space, and could affect the results of some `ORDER BY` queries that made use of such indexes.

  Note

  When upgrading, it is necessary to re-create any existing indexes on Unicode `CHAR` columns of each affected table to take advantage of the fix. See Section 2.11.4, “Rebuilding or Repairing Tables or Indexes”.

  (Bug #22052)
- Incorrect results could be returned for some queries that contained a select list expression with `IN` or `BETWEEN` [783] together with an `ORDER BY` or `GROUP BY` on the same expression using `NOT IN` or `NOT BETWEEN`. (Bug #27532)
- Index hints (`USE INDEX`, `IGNORE INDEX`, `FORCE INDEX`) cannot be used with `FULLTEXT` indexes, but were not being ignored. (Bug #25951)
- Changes to some system variables should invalidate statements in the query cache, but invalidation did not happen. (Bug #27792)
- Queries using a column alias in an expression as part of an `ORDER BY` clause failed, an example of such a query being `SELECT mycol + 1 AS mynum FROM mytable ORDER BY 30 - mynum`. (Bug #22457)
- The range optimizer could consume a combinatorial amount of memory for certain classes of `WHERE` clauses. (Bug #26624)
- Attempts to access a `MyISAM` table with a corrupt column definition caused a server crash. (Bug #24401)
- The `InnoDB` parser sometimes did not account for null bytes, causing spurious failure of some queries. (Bug #25596)
- Storing `NULL` values in spatial fields caused excessive memory allocation and crashes on some systems. (Bug #27164)
- Optimizations that are legal only for subqueries without tables and `WHERE` conditions were applied for any subquery without tables. (Bug #24670)
- `mysqltest` crashed with a stack overflow. (Bug #24498)
- `ALTER TABLE` statements that performed both `RENAME TO` and `{ENABLE|DISABLE} KEYS` operations caused a server crash. (Bug #24219)
- In a `MEMORY` table, using a `BTREE` index to scan for updatable rows could lead to an infinite loop. (Bug #26996)
- The range optimizer could cause the server to run out of memory. (Bug #26625)

- Storing values specified as hexadecimal values 64 or more bits long into `BIGINT` or `BIGINT UNSIGNED` columns did not raise any warning or error if the value was out of range. (Bug #22533)
- The number of `setsockopt()` calls performed for reads and writes to the network socket was reduced to decrease system call overhead. (Bug #22943)
- `mysql` did not check for errors when fetching data during result set printing. (Bug #22913)
- Changing the value of `MI_KEY_BLOCK_LENGTH` in `myisam.h` and recompiling MySQL resulted in a `myisamchk` that saw existing `MyISAM` tables as corrupt. (Bug #22119)
- `IN()` [784] and `CHAR()` [794] can return `NULL`, but did not signal that to the query processor, causing incorrect results for `IS NULL` [783] operations. (Bug #17047)
- For `ALTER TABLE`, using `ORDER BY` *`expression`* could cause a server crash. Now the `ORDER BY` clause permits only column names to be specified as sort criteria (which was the only documented syntax, anyway). (Bug #24562)
- `ORDER BY` values of the `DOUBLE` or `DECIMAL` types could change the result returned by a query. (Bug #19690)
- Hebrew-to-Unicode conversion failed for some characters. Definitions for the following Hebrew characters (as specified by the ISO/IEC 8859-8:1999) were added: LEFT-TO-RIGHT MARK (LRM), RIGHT-TO-LEFT MARK (RLM) (Bug #24037)
- User-defined variables could consume excess memory, leading to a crash caused by the exhaustion of resources available to the `MEMORY` storage engine, due to the fact that this engine is used by MySQL for variable storage and intermediate results of `GROUP BY` queries. Where `SET` had been used, such a condition could instead give rise to the misleading error message `You may only use constant expressions with SET`, rather than `Out of memory (Needed NNNNNN bytes)`. (Bug #23443)
- A table created with the `ROW_FORMAT = FIXED` table option lost that option if an index was added or dropped with `CREATE INDEX` or `DROP INDEX`. (Bug #23404)
- Difficult repair or optimization operations could cause an assertion failure, resulting in a server crash. (Bug #25289)
- The stack size for NetWare binaries was increased to 128KB to prevent problems caused by insufficient stack size. (Bug #23504)
- `InnoDB`: During a restart of the MySQL Server that followed the creation of a temporary table using the `InnoDB` storage engine, MySQL failed to clean up in such a way that `InnoDB` still attempted to find the files associated with such tables. (Bug #20867)
- Foreign key identifiers for `InnoDB` tables could not contain certain characters. (Bug #24299)
- Some long error messages were printed incorrectly. (Bug #20710)
- Conversion of `DATETIME` values in numeric contexts sometimes did not produce a double (`YYYYMMDDHHMMSS.uuuuuu`) value. (Bug #16546)
- Comparisons using row constructors could fail for rows containing `NULL` values. (Bug #27704)
- Through the C API, the member strings in `MYSQL_FIELD` for a query that contained expressions could return incorrect results. (Bug #21635)
- Range searches on columns with an index prefix could miss records. (Bug #20732)
- `perror` crashed on some platforms due to failure to handle a `NULL` pointer. (Bug #25344)

- `mysql` would lose its connection to the server if its standard output was not writable. (Bug #17583)
- `INSERT...ON DUPLICATE KEY UPDATE` could cause `Error 1032: Can't find record in ...` for inserts into an `InnoDB` table unique index using key column prefixes with an underlying `utf8` string column. (Bug #13191)
- In certain cases it could happen that deleting a row corrupted an `RTREE` index. This affected indexes on spatial columns. (Bug #25673)
- The server was built even when `configure` was run with the `--without-server` [95] option. (Bug #23973)

  References: See also Bug #32898.
- `ALTER TABLE ENABLE KEYS` or `ALTER TABLE DISABLE KEYS` combined with another `ALTER TABLE` option other than `RENAME TO` did nothing. In addition, if `ALTER TABLE` was used on a table having disabled keys, the keys of the resulting table were enabled. (Bug #24395)
- Certain joins using `Range checked for each record` in the query execution plan could cause the server to crash. (Bug #24776)
- The server might fail to use an appropriate index for `DELETE` when `ORDER BY`, `LIMIT`, and a nonrestricting `WHERE` are present. (Bug #17711)
- Adding a day, month, or year interval to a `DATE` value produced a `DATE`, but adding a week interval produced a `DATETIME` value. Now all produce a `DATE` value. (Bug #21811)

C.1.4 Changes in MySQL 4.1.22 (2006-11-02)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.1 release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details, please see http://www.mysql.com/products/enterprise.

Functionality Added or Changed

- The `LOAD DATA FROM MASTER` and `LOAD TABLE FROM MASTER` statements are deprecated. See Section 12.5.2.2, “`LOAD DATA FROM MASTER` Syntax”, for recommended alternatives. (Bug #9125, Bug #20596, Bug #14399, Bug #12187, Bug #15025, Bug #18822)
- The `mysqld` man page has been reclassified from volume 1 to volume 8. (Bug #21220)
- MySQL did not properly do stack dumps on `x86_64` and `i386/NPTL` systems. (Note that the initial fix for this problem was discovered not to be correct. Further work on the problem was undertaken only for MySQL 5.1 and up. See Bug #31891.) (Bug #21250)
- A warning now is issued if the client attempts to set the `sql_log_off` [429] variable without the `SUPER` [493] privilege. (Bug #16180)

Bugs Fixed

- **Packaging; MySQL Cluster:** The `ndb_mgm` program was included in both the `MySQL-ndb-tools` and `MySQL-ndb-management` RPM packages, resulting in a conflict if both were installed. Now `ndb_mgm` is included only in `MySQL-ndb-tools`. (Bug #21058)
- **MySQL Cluster:** When inserting a row into an `NDB` table with a duplicate value for a nonprimary unique key, the error issued would reference the wrong key. (Bug #21072)
- **MySQL Cluster:** In some situations with a high disk-load, writing of the redo log could hang, causing a crash with the error message `GCP STOP detected`. (Bug #20904)
- **MySQL Cluster:** Multiple node restarts in rapid succession could cause a system restart to fail , or induce a race condition. (Bug #22892, Bug #23210)
- **MySQL Cluster:** The output for the `--help` option used with `NDB` executable programs (such as `ndbd`, `ndb_mgm`, `ndb_restore`, `ndb_config`, and others mentioned in Section 15.4, "MySQL Cluster Programs") referred to the `Ndb.cfg` file, instead of to `my.cnf`. (Bug #21585)
- **MySQL Cluster:** `ndb_size.pl` and `ndb_error_reporter` were missing from RPM packages. (Bug #20426)
- **MySQL Cluster:** The failure of a unique index read due to an invalid schema version could be handled incorrectly in some cases, leading to unpredictable results. (Bug #21384)
- **MySQL Cluster:** The `ndb_mgm` management client did not set the exit status on errors, always returning 0 instead. (Bug #21530)
- **MySQL Cluster:** Under some circumstances, local checkpointing would hang, keeping any unstarted nodes from being started. (Bug #20895)
- **MySQL Cluster:** Setting `TransactionDeadlockDetectionTimeout` [1269] to a value greater than 12000 would cause scans to deadlock, time out, fail to release scan records, until the cluster ran out of scan records and stopped processing. (Bug #21800)
- **MySQL Cluster:** Some queries involving joins on very large `NDB` tables could crash the MySQL server. (Bug #21059)
- **MySQL Cluster:** A partial rollback could lead to node restart failures. (Bug #21536)
- **MySQL Cluster:** If a node restart could not be performed from the REDO log, no node takeover took place. This could cause partitions to be left empty during a system restart. (Bug #22893)
- **MySQL Cluster:** The `ndb_size.pl` script did not account for `TEXT` and `BLOB` column values correctly. (Bug #21204)
- **MySQL Cluster:** Attempting to create an `NDB` table on a MySQL with an existing non-Cluster table with the same name in the same database could result in data loss or corruption. MySQL now issues a warning when a `SHOW TABLES` or other statement causing table discovery finds such a table. (Bug #21378)
- **MySQL Cluster:** `INSERT ... ON DUPLICATE KEY UPDATE` on an `NDB` table could lead to deadlocks and memory leaks. (Bug #23200)
- **MySQL Cluster:** The server provided a nondescriptive error message when encountering a fatally corrupted REDO log. (Bug #21615)
- **MySQL Cluster:** `ndb_restore` did not always make clear that it had recovered successfully from temporary errors while restoring a cluster backup. (Bug #19651)

- **MySQL Cluster:** Cluster logs were not rotated following the first rotation cycle. (Bug #21345)
- **MySQL Cluster:** When the redo buffer ran out of space, a `Pointer too large` error was raised and the cluster could become unusable until restarted with `--initial`. (Bug #20892)
- **MySQL Cluster:** Backup of a cluster failed if there were any tables with 128 or more columns. (Bug #23502)
- **MySQL Cluster:** A problem with takeover during a system restart caused ordered indexes to be rebuilt incorrectly. (Bug #15303)
- **MySQL Cluster:** In some cases where `SELECT COUNT(*)` from an `NDB` table should have yielded an error, `MAX_INT` was returned instead. (Bug #19914)
- **MySQL Cluster:** (NDB API): Attempting to read a nonexistent tuple using `Commit` mode for `NdbTransaction::execute()` caused node failures. (Bug #22672)
- **MySQL Cluster:** `SELECT ... FOR UPDATE` failed to lock the selected rows. (Bug #18184)
- **MySQL Cluster:** The node recovery algorithm was missing a version check for tables in the `ALTER_TABLE_COMMITTED` state (as opposed to the `TABLE_ADD_COMMITTED` state, which has the version check). This could cause inconsistent schemas across nodes following node recovery. (Bug #21756)
- **MySQL Cluster:** In a cluster with more than 2 replicas, a manual restart of one of the data nodes could fail and cause the other nodes in the same node group to shut down. (Bug #21213)
- **MySQL Cluster:** The server failed with a nondescriptive error message when out of data memory. (Bug #18475)
- Using `ALTER TABLE` to add an `ENUM` column with an enumeration value containing `0xFF` caused the name of the first table column to be lost. (Bug #20922)
- `SUBSTRING()` [802] results sometimes were stored improperly into a temporary table when multi-byte character sets were used. (Bug #20204)
- `mysql_install_db` incorrectly had an empty first line. (Bug #20721)
- The optimizer could produce an incorrect result after `AND` with collations such as `latin1_german2_ci`, `utf8_czech_ci`, and `utf8_lithuanian_ci`. (Bug #9509)
- If a column definition contained a character set declaration, but a `DEFAULT` value began with an introducer, the introducer character set was used as the column character set. (Bug #20695)
- User names have a maximum length of 16 characters (even if they contain multi-byte characters), but were being truncated to 16 bytes. (Bug #20393)
- `PROCEDURE ANALYSE()` returned incorrect values of *`M`* `FLOAT(`*`M, D`*`)` and `DOUBLE(`*`M, D`*`)`. (Bug #20305)
- For a `MyISAM` table with a `FULLTEXT` index, compression with `myisampack` or a check with `myisamchk` after compression resulted in table corruption. (Bug #19702)
- A query using `WHERE` *`column = constant`* `OR` *`column`* `IS NULL` did not return consistent results on successive invocations. The *`column`* in each part of the `WHERE` clause could be either the same column, or two different columns, for the effect to be observed. (Bug #21019)
- `mysqld --flush` failed to flush `MyISAM` table changes to disk following an `UPDATE` statement for which no updated column had an index. (Bug #20060)

- For `TIME_FORMAT()` [840], the `%H` and `%k` format specifiers can return values larger than two digits (if the hour is greater than 99), but for some query results that contained three-character hours, column values were truncated. (Bug #19844)
- A subquery that uses an index for both the `WHERE` and `ORDER BY` clauses produced an empty result. (Bug #21180)
- Some Linux-x86_64-icc packages (of previous releases) mistakenly contained 32-bit binaries. Only `ICC` builds are affected, not `gcc` builds. Solaris and FreeBSD x86_64 builds are not affected. (Bug #22238)
- Redundant binary log `LAST_INSERT_ID` events could be generated; `LAST_INSERT_ID(`*`expr`*`)` [874] did not return the value of *`expr`*; `LAST_INSERT_ID()` [874] could return the value generated by the current statement if the call occurred after value generation, as in:

```
CREATE TABLE t1 (i INT AUTO_INCREMENT PRIMARY KEY, j INT);
INSERT INTO t1 VALUES (NULL, 0), (NULL, LAST_INSERT_ID());
```

  (Bug #21726)
- A query that used `GROUP BY` and an `ALL` or `ANY` quantified subquery in a `HAVING` clause could trigger an assertion failure. (Bug #21853)
- `EXPORT_SET()` [795] did not accept arguments with coercible character sets. (Bug #21531)
- The source distribution failed to compile when configured with the `--without-geometry` option. (Bug #12991)
- For `INSERT ... ON DUPLICATE KEY UPDATE`, use of `VALUES(`*`col_name`*`)` [880] within the `UPDATE` clause sometimes was handled incorrectly. (Bug #21555)
- When using tables containing `VARCHAR` columns created under MySQL 4.1 with a 5.0 or later server, for some queries the metadata sent to the client could have an empty column name. (Bug #14897)
- Incorporated portability fixes into the definition of `__attribute__` in `my_global.h`. (Bug #2717)
- Under heavy load (executing more than 1024 simultaneous complex queries), a problem in the code that handles internal temporary tables could lead to writing beyond allocated space and memory corruption. (Bug #21206)
- The `--collation-server` [385] server option was being ignored. With the fix, if you choose a nondefault character set with `--character-set-server` [385], you should also use `--collation-server` [385] to specify the collation. (Bug #15276)
- On Mac OS X, zero-byte `read()` or `write()` calls to an SMB-mounted file system could return a nonstandard return value, leading to data corruption. Now such calls are avoided. (Bug #12620)
- `LIKE` searches failed for indexed `utf8` character columns. (Bug #20471)
- The `MD5()` [868], `SHA1()` [869], and `ENCRYPT()` [867] functions should return a binary string, but the result sometimes was converted to the character set of the argument. `MAKE_SET()` [799] and `EXPORT_SET()` [795] now use the correct character set for their default separators, resulting in consistent result strings which can be coerced according to normal character set rules. (Bug #20536)
- Certain malformed `INSERT` statements could crash the `mysql` client. (Bug #21142)
- Entries in the slow query log could have an incorrect `Rows_examined` value. (Bug #12240)

- The result for `CAST()` [859] when casting a value to `UNSIGNED` was limited to the maximum signed `BIGINT` value (9223372036854775808), rather than the maximum unsigned value (18446744073709551615). (Bug #8663)
- Using the extended syntax for `TRIM()` [803]—that is, `TRIM(... FROM ...)` [803]—in a `SELECT` statement defining a view caused an invalid syntax error when selecting from the view. (Bug #17526)
- `OPTIMIZE TABLE` with `myisam_repair_threads` [421] > 1 could result in `MyISAM` table corruption. (Bug #8283)
- `WITH ROLLUP` could group unequal values. (Bug #20825)
- `REPAIR TABLE ... USE_FRM` could cause a server crash or hang when used for a `MyISAM` table in a database other than the default database. (Bug #22562)
- Insufficient memory (`myisam_sort_buffer_size` [421]) could cause a server crash for several operations on `MyISAM` tables: repair table, create index by sort, repair by sort, parallel repair, bulk insert. (Bug #23175)
- Execution of a prepared statement that uses an `IN` subquery with aggregate functions in the `HAVING` clause could cause a server crash. (Bug #22085)
- The `myisam_stats_method` [422] variable was mishandled when set from an option file or on the command line. (Bug #21054)
- Using `ANY` with “nontable” subqueries such as `SELECT 1` yielded incorrect results under certain circumstances due to incorrect application of `MIN()` [884]/`MAX()` [884] optimization. (Bug #16302)
- For cross-database multiple-table `UPDATE` statements, a user with all privileges for the default database could update tables in another database for which the user did not have `UPDATE` [493] privileges. (Bug #7391)
- Adding `ORDER BY` to a `SELECT DISTINCT(expr)` query could produce incorrect results. (Bug #21456)
- `COUNT(*)` [882] queries with `ORDER BY` and `LIMIT` could return the wrong result.

  Note

  This problem was introduced by the fix for Bug #9676, which limited the rows stored in a temporary table to the `LIMIT` clause. This optimization is not applicable to nongroup queries with aggregate functions. The current fix disables the optimization in such cases.

  (Bug #21787)
- Conversion of `TIMESTAMP` values between UTC and the local time zone resulted in some values having the year 2069 rather than 1969. (Bug #16327)
- `DELETE IGNORE` could hang for foreign key parent deletes. (Bug #18819)
- A query using `WHERE NOT (column < ANY (subquery))` yielded a different result from the same query using the same `column` and `subquery` with `WHERE (column > ANY (subquery))`. (Bug #20975)
- Creating a `TEMPORARY` table with the same name as an existing table that was locked by another client could result in a lock conflict for `DROP TEMPORARY TABLE` because the server unnecessarily tried to acquire a name lock. (Bug #21096)

- `FROM_UNIXTIME()` [834] did not accept arguments up to `POWER(2,31)-1` [822], which it had previously. (Bug #9191)
- `libmysqld` returned `TEXT` columns to the client as number of bytes, not number of characters (which can be different for multi-byte character sets). (Bug #19983)
- A literal string in a `GROUP BY` clause could be interpreted as a column name. (Bug #14019)
- The `--with-collation` [97] option was not honored for client connections. (Bug #7192)
- A patch fixing the omission of leading zeros in dates in MySQL 4.1.21 was reverted.

  References: The patch for the following bug was reverted: Bug #16377.
- Multiple invocations of the `REVERSE()` [800] function could return different results. (Bug #18243)
- Within a prepared statement, `SELECT (COUNT(*) = 1)` (or similar use of other aggregate functions) did not return the correct result for statement re-execution. (Bug #21354)
- Running `SHOW MASTER LOGS` at the same time as binary log files were being switched would cause `mysqld` to hang. (Bug #21965)
- For multiple-table `UPDATE` statements, storage engines were not notified of duplicate-key errors. (Bug #21381)
- A subquery in the `WHERE` clause of the outer query and using `IN` and `GROUP BY` returned an incorrect result. (Bug #16255)
- A server or network failure with an open client connection would cause the client to hang even though the server was no longer available.

  As a result of this change, the `MYSQL_OPT_READ_TIMEOUT` and `MYSQL_OPT_WRITE_TIMEOUT` options for `mysql_options()` now apply to TCP/IP connections on all platforms. Previously, they applied only to Windows. (Bug #9678)
- `DELETE` with `WHERE` condition on a `BTREE`-indexed column for a `MEMORY` table deleted only the first matched row. (Bug #9719)
- Using aggregate functions in subqueries yielded incorrect results under certain circumstances due to incorrect application of `MIN()` [884]/`MAX()` [884] optimization. (Bug #20792)
- Deleting entries from a large `MyISAM` index could cause index corruption when it needed to shrink. Deletes from an index can happen when a record is deleted, when a key changes and must be moved, and when a key must be un-inserted because of a duplicate key. This can also happen in `REPAIR TABLE` when a duplicate key is found and in `myisamchk` when sorting the records by an index. (Bug #22384)
- For an `ENUM` column that used the `ucs2` character set, using `ALTER TABLE` to modify the column definition caused the default value to be lost. (Bug #20108)
- The server returns a more informative error message when it attempts to open a `MERGE` table that has been defined to use non-`MyISAM` tables. (Bug #10974)
- `libmysqld` produced some warnings to `stderr` which could not be silenced. These warnings now are suppressed. (Bug #13717)
- Character set collation was ignored in `GROUP BY` clauses. (Bug #20709)
- Use of the join cache in favor of an index for `ORDER BY` operations could cause incorrect result sorting. (Bug #17212)

- The optimizer sometimes mishandled R-tree indexes for `GEOMETRY` data types, resulting in a server crash. (Bug #21888)
- Views could not be updated within a stored function or trigger. (Bug #17591)
- Setting `myisam_repair_threads` [421] caused any repair operation on a `MyISAM` table to fail to update the cardinality of indexes, instead making them always equal to 1. (Bug #18874)
- Table aliases in multiple-table `DELETE` statements sometimes were not resolved. (Bug #21392)
- Parallel builds occasionally failed on Solaris. (Bug #16282)
- On 64-bit systems, use of the `cp1250` character set with a primary key column in a `LIKE` clause caused a server crash for patterns having letters in the range 128..255. (Bug #19741)
- The use of `WHERE col_name IS NULL` in `SELECT` statements reset the value of `LAST_INSERT_ID()` [874] to zero. (Bug #14553)
- For table-format output, `mysql` did not always calculate columns widths correctly for columns containing multi-byte characters in the column name or contents. (Bug #17939)
- The build process incorrectly tried to overwrite `sql/lex_hash.h`. This caused the build to fail when using a shadow link tree pointing to original sources that were owned by another account. (Bug #18888)
- Selecting from a `MERGE` table could result in a server crash if the underlying tables had fewer indexes than the `MERGE` table itself. (Bug #21617, Bug #22937)
- `character_set_results` [408] can be `NULL` to signify “no conversion,” but some code did not check for `NULL`, resulting in a server crash. (Bug #21913)
- Using `> ALL` with subqueries that return no rows yielded incorrect results under certain circumstances due to incorrect application of `MIN()` [884]/`MAX()` [884] optimization. (Bug #18503)
- For `InnoDB` tables, the server could crash when executing `NOT IN(...)` subqueries. (Bug #21077)
- Under certain circumstances, `AVG(key_val)` [881] returned a value but `MAX(key_val)` [884] returned an empty set due to incorrect application of `MIN()/MAX()` [884] optimization. (Bug #20954)
- In the package of pre-built time zone tables that is available for download at http://dev.mysql.com/downloads/timezones.html, the tables now explicitly use the `utf8` character set so that they work the same way regardless of the system character set value. (Bug #21208)
- Queries containing a subquery that used aggregate functions could return incorrect results. (Bug #16792)

C.1.5 Changes in MySQL 4.1.21 (2006-07-19)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.1 release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details, please see http://www.mysql.com/products/enterprise.

Functionality Added or Changed

- For a table with an `AUTO_INCREMENT` column, `SHOW CREATE TABLE` now shows the next `AUTO_INCREMENT` value to be generated. (Bug #19025)
- The `mysqldumpslow` script has been moved from client RPM packages to server RPM packages. This corrects a problem where `mysqldumpslow` could not be used with a client-only RPM install, because it depends on `my_print_defaults` which is in the server RPM. (Bug #20216)
- Added the `--set-charset` [298] option to `mysqlbinlog` to enable the character set to be specified for processing binary log files. (Bug #18351)
- For spatial data types, the server formerly returned these as `VARSTRING` values with a binary collation. Now the server returns spatial values as `BLOB` values. (Bug #10166)
- A new system variable, `lc_time_names` [417], specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()` [832], `DAYNAME()` [833] and `MONTHNAME()` [837] functions. See Section 9.8, “MySQL Server Locale Support”.

Bugs Fixed

- **Security Fix:** If a user has access to `MyISAM` table *`t`*, that user can create a `MERGE` table *`m`* that accesses *`t`*. However, if the user's privileges on *`t`* are subsequently revoked, the user can continue to access *`t`* by doing so through *`m`*. If this behavior is undesirable, you can start the server with the new `--skip-merge` [393] option to disable the `MERGE` storage engine. (Bug #15195, CVE-2006-4031)
- **Security Fix:** Invalid arguments to `DATE_FORMAT()` [832] caused a server crash. Thanks to Jean-David Maillefer for discovering and reporting this problem to the Debian project and to Christian Hammers from the Debian Team for notifying us of it. (Bug #20729, CVE-2006-3469)
- **Security Fix:** On Linux, and possibly other platforms using case-sensitive file systems, it was possible for a user granted rights on a database to create or access a database whose name differed only from that of the first by the case of one or more letters. (Bug #17647, CVE-2006-4226)
- **MySQL Cluster:** Resources for unique indexes on Cluster table columns were incorrectly allocated, so that only one-fourth as many unique indexes as indicated by the value of `UniqueHashIndexes` could be created. (Bug #19623)
- **MySQL Cluster:** It was possible to use port numbers greater than 65535 for `ServerPort` [1249] in the `config.ini` file. (Bug #19164)
- **MySQL Cluster:** Repeated use of the `SHOW` and `ALL STATUS` commands in the `ndb_mgm` client could cause the `mgmd` process to crash. (Bug #18591)
- **MySQL Cluster:** Renaming a table in such a way as to move it to a different database failed to move the table's indexes. (Bug #19967)
- **MySQL Cluster:** Using “stale” `mysqld .frm` files could cause a newly restored cluster to fail. This situation could arise when restarting a MySQL Cluster using the `--initial` option while leaving connected `mysqld` processes running. (Bug #16875)
- **MySQL Cluster:** A problem with error handling when `ndb_use_exact_count` [1305] was enabled could lead to incorrect values returned from queries using `COUNT()` [882]. A warning is now returned in such cases. (Bug #19202)
- **MySQL Cluster:** A Cluster whose storage nodes were installed from the `MySQL-ndb-storage-*` RPMs could not perform `CREATE` or `ALTER` operations that made use of nondefault character sets or collations. (Bug #14918)

- **MySQL Cluster:** The failure of a data node when preparing to commit a transaction (that is, while the node's status was `CS_PREPARE_TO_COMMIT`) could cause the failure of other cluster data nodes. (Bug #20185)
- **MySQL Cluster:** Data node failures could cause excessive CPU usage by `ndb_mgmd`. (Bug #13987)
- **MySQL Cluster:** A node failure during a scan could sometime cause the node to crash when restarting too quickly following the failure. (Bug #20197)
- **MySQL Cluster:** Some queries having a `WHERE` clause of the form `c1=val1 OR c2 LIKE 'val2'` were not evaluated correctly. (Bug #17421)
- **MySQL Cluster:** `TRUNCATE TABLE` failed on tables having `BLOB` or `TEXT` columns with the error `Lock wait timeout exceeded`. (Bug #19201)
- **MySQL Cluster:** An issue with `ndb_mgmd` prevented more than 27 `mysqld` processes from connecting to a single cluster at one time. (Bug #17150)
- **MySQL Cluster:** The `ndb_mgm` client command `ALL CLUSTERLOG STATISTICS=15` had no effect. (Bug #20336)
- **MySQL Cluster:** `LOAD DATA LOCAL` failed to ignore duplicate keys in Cluster tables. (Bug #19496)
- **MySQL Cluster:** The repeated creating and dropping of a table would eventually lead to `NDB` Error 826, `Too many tables and attributes ... Insufficient space`. (Bug #20847)
- **MySQL Cluster:** The cluster's data nodes failed while trying to load data when `NoOfFrangmentLogFiles` was set equal to 1. (Bug #19894)
- **MySQL Cluster:** `TRUNCATE TABLE` failed to reset the `AUTO_INCREMENT` counter. (Bug #18864)
- **MySQL Cluster:** When attempting to restart the cluster following a data import, the cluster failed during Phase 4 of the restart with `Error 2334: Job buffer congestion`. (Bug #20774)
- **MySQL Cluster:** Repeated `CREATE` - `INSERT` - `DROP` operations on tables could in some circumstances cause the MySQL table definition cache to become corrupt, so that some `mysqld` processes could access table information but others could not. (Bug #18595)
- **Replication:** The binary log would create an incorrect `DROP` query when creating temporary tables during replication. (Bug #17263)
- **Cluster API:** On big-endian platforms, `NdbOperation::write_attr()` did not update 32-bit fields correctly. (Bug #19537)
- Checking a `MyISAM` table (using `CHECK TABLE`) having a spatial index and only one row would wrongly indicate that the table was corrupted. (Bug #17877)
- Use of `MIN()` [884] or `MAX()` [884] with `GROUP BY` on a `ucs2` column could cause a server crash. (Bug #20076)
- Multiple-table `DELETE` statements containing a subquery that selected from one of the tables being modified caused a server crash. (Bug #19225)
- Concatenating the results of multiple constant subselects produced incorrect results. (Bug #16716)
- `ANALYZE TABLE` for `TEMPORARY` tables had no effect. (Bug #15225)
- The `fill_help_tables.sql` file did not contain a `SET NAMES 'utf8'` statement to indicate its encoding. This caused problems for some settings of the MySQL character set such as `big5`. (Bug #20551)

- The binary log lacked character set information for table names when dropping temporary tables. (Bug #14157)
- `mysqldump` did not respect the order of tables named with the `--tables` [299] option. (Bug #18536)
- For a reference to a nonexistent index in `FORCE INDEX`, the error message referred to a column, not an index. (Bug #17873)
- `DATE_ADD()` [829] and `DATE_SUB()` [833] returned `NULL` when the result date was on the day `'9999-12-31'`. (Bug #12356)
- The `DATA DIRECTORY` table option did not work for `TEMPORARY` tables. (Bug #8706)
- The `ARCHIVE` storage engine does not support `TRUNCATE TABLE`, but the server was not returning an appropriate error when truncation of an `ARCHIVE` table was attempted. (Bug #15558)
- Improper character set initialization in the embedded server could result in a server crash. (Bug #20318)
- For a `DATE` parameter sent using a `MYSQL_TIME` data structure, `mysql_stmt_execute()` zeroed the hour, minute, and second members of the structure rather than treating them as read only. (Bug #20152)
- Certain queries having a `WHERE` clause that included conditions on multi-part keys with more than 2 key parts could produce incorrect results and send `[Note] Use_count: Wrong count for key at...` messages to `STDERR`. (Bug #16168)
- `InnoDB` unlocked its data directory before committing a transaction, potentially resulting in nonrecoverable tables if a server crash occurred before the commit. (Bug #19727)
- Invalid escape sequences in option files caused MySQL programs that read them to abort. (Bug #15328)
- Queries using an indexed column as the argument for the `MIN()` [884] and `MAX()` [884] functions following an `ALTER TABLE .. DISABLE KEYS` statement returned `Got error 124 from storage engine` until `ALTER TABLE ... ENABLE KEYS` was run on the table. (Bug #20357)
- For very complex `SELECT` statements could create temporary tables that were too large, and for which the temporary files were not removed, causing subsequent queries to fail. (Bug #11824)
- For `SELECT ... FOR UPDATE` statements that used `DISTINCT` or `GROUP BY` over all key parts of a unique index (or primary key), the optimizer unnecessarily created a temporary table, thus losing the linkage to the underlying unique index values. This caused a `Result set not updatable` error. (The temporary table is unnecessary because under these circumstances the distinct or grouped columns must also be unique.) (Bug #16458)
- `IS_USED_LOCK()` [879] could return an incorrect connection identifier. (Bug #16501)
- The server no longer uses a signal handler for signal 0 because it could cause a crash on some platforms. (Bug #15869)
- A statement containing `GROUP BY` and `HAVING` clauses could return incorrect results when the `HAVING` clause contained logic that returned `FALSE` for every row. (Bug #14927)
- The use of `MIN()` [884] and `MAX()` [884] on columns with an index prefix produced incorrect results in some queries. (Bug #18206)
- `INSERT INTO ... SELECT ... LIMIT 1` could be slow because the `LIMIT` was ignored when selecting candidate rows. (Bug #9676)
- `InnoDB` failed to increment the `handler_read_prev` counter. (Bug #19542)

- An invalid comparison between keys with index prefixes over multi-byte character fields could lead to incorrect result sets if the selected query execution plan used a range scan by an index prefix over a `UTF8` character field. This also caused incorrect results under similar circumstances with many other character sets. (Bug #14896)
- Closing of temporary tables failed if binary logging was not enabled. (Bug #20919)
- An update that used a join of a table to itself and modified the table on both sides of the join reported the table as crashed. (Bug #18036)
- The `MD5()` [868] and `SHA()` [869] functions treat their arguments as case-sensitive strings. But when they are compared, their arguments were compared as case-insensitive strings, which leads to two function calls with different arguments (and thus different results) compared as being identical. This can lead to a wrong decision made in the range optimizer and thus to an incorrect result set. (Bug #15351)
- Using `SELECT` and a table join while running a concurrent `INSERT` operation would join incorrect rows. (Bug #14400)
- The `fill_help_tables.sql` file did not load properly if the `ANSI_QUOTES` [458] SQL mode was enabled. (Bug #20542)
- The MySQL server startup script `/etc/init.d/mysql` (created from `mysql.server`) is now marked to ensure that the system services `ypbind`, `nscd`, `ldap`, and `NTP` are started first (if these are configured on the machine). (Bug #18810)
- The `ref` [567] optimizer could choose the `ref_or_null` [567] access method in cases where it was not applicable. This could cause inconsistent `EXPLAIN` or `SELECT` results for a given statement. (Bug #16798)
- The `mysql` client did not understand `help` commands that had spaces at the end. (Bug #20328)
- Concurrent reading and writing of privilege structures could crash the server. (Bug #16372)
- Slave SQL thread cleanup was not handled properly on Mac OS X when a statement was killed, resulting in a slave crash. (Bug #16900)
- When `mysqldump` disabled keys and locked a `MyISAM` table, the lock operation happened second. If another client performed a query on the table in the interim, it could take a long time due to indexes not being used. Now the lock operation happens first. (Bug #15977)
- `LOAD_FILE()` [798] returned an error if the file did not exist, rather than `NULL` as it should according to the manual. (Bug #10418)
- `SHOW CREATE TABLE` did not display the `AUTO_INCREMENT` column attribute if the SQL mode was `MYSQL323` [460] or `MYSQL40` [461]. This also affected `mysqldump`, which uses `SHOW CREATE TABLE` to get table definitions. (Bug #14515)
- The `mysql` client did not ignore client-specific commands (such as `use` or `help`) that occurred as the first word on a line within multiple-line `/* ... */` comments. (Bug #20432)
- A number of dependency issues in the RPM `bench` and `test` packages caused installation of these packages to fail. (Bug #20078)
- In a multiple-row `INSERT` statement, `LAST_INSERT_ID()` [874] should return the same value for each row. However, in some cases, the value could change if the table being inserted into had its own `AUTO_INCREMENT` column. (Bug #6880)
- Some memory leaks in the `libmysqld` embedded server were corrected. (Bug #16017)

- Some queries that used `ORDER BY` and `LIMIT` performed quickly in MySQL 3.23, but slowly in MySQL 4.x/5.x due to an optimizer problem. (Bug #4981)
- `MONTHNAME(STR_TO_DATE(NULL, '%m'))` [837] could cause a server crash. (Bug #18501)
- The omission of leading zeros in dates could lead to erroneous results when these were compared with the output of certain date and time functions.

  Note

  The patch for this bug was reverted in MySQL 4.1.22.

  (Bug #16377)
- Repeated `DROP TABLE` statements in a stored procedure could sometimes cause the server to crash. (Bug #19399)
- The length of the pattern string prefix for `LIKE` operations was calculated incorrectly for multi-byte character sets. As a result, the scanned range was wider than necessary if the prefix contained any multi-byte characters, and rows could be missing from the result set. (Bug #18359, Bug #16674)
- Using `SELECT` on a corrupt `MyISAM` table using the dynamic record format could cause a server crash. (Bug #19835)
- No error message was being issued for storage engines that do not support `ALTER TABLE`. Now an `ER_NOT_SUPPORTED_YET` [1583] error occurs. (Bug #7643)
- A cast problem caused incorrect results for prepared statements that returned float values when MySQL was compiled with `gcc` 4.0. (Bug #19694)
- Use of uninitialized user variables in a subquery in the `FROM` clause resulted in invalid entries in the binary log. (Bug #19136)
- When `myisamchk` needed to rebuild a table, `AUTO_INCREMENT` information was lost. (Bug #10405)
- Failure to account for a `NULL` table pointer on big-endian machines could cause a server crash during type conversion. (Bug #21135)

C.1.6 Changes in MySQL 4.1.20 (2006-05-24)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a security fix release for the MySQL 4.1 release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details, please see http://www.mysql.com/products/enterprise.

Bugs Fixed

- **Security Fix:** An SQL-injection security hole has been found in multi-byte encoding processing. The bug was in the server, incorrectly parsing the string escaped with the `mysql_real_escape_string()` C API function.

This vulnerability was discovered and reported by Josh Berkus <josh@postgresql.org> and Tom Lane <tgl@sss.pgh.pa.us> as part of the inter-project security collaboration of the OSDB consortium. For more information about SQL injection, please see the following text.

Discussion. An SQL injection security hole has been found in multi-byte encoding processing. An SQL injection security hole can include a situation whereby when a user supplied data to be inserted into a database, the user might inject SQL statements into the data that the server will execute. With regards to this vulnerability, when character set-unaware escaping is used (for example, `addslashes()` in PHP), it is possible to bypass the escaping in some multi-byte character sets (for example, SJIS, BIG5 and GBK). As a result, a function such as `addslashes()` is not able to prevent SQL-injection attacks. It is impossible to fix this on the server side. The best solution is for applications to use character set-aware escaping offered by a function such `mysql_real_escape_string()`.

However, a bug was detected in how the MySQL server parses the output of `mysql_real_escape_string()`. As a result, even when the character set-aware function `mysql_real_escape_string()` was used, SQL injection was possible. This bug has been fixed.

Workarounds. If you are unable to upgrade MySQL to a version that includes the fix for the bug in `mysql_real_escape_string()` parsing, but run MySQL 5.0.1 or higher, you can use the `NO_BACKSLASH_ESCAPES` SQL mode as a workaround. (This mode was introduced in MySQL 5.0.1.) `NO_BACKSLASH_ESCAPES` enables an SQL standard compatibility mode, where backslash is not considered a special character. The result will be that queries will fail.

To set this mode for the current connection, enter the following SQL statement:

```
SET sql_mode='NO_BACKSLASH_ESCAPES';
```

You can also set the mode globally for all clients:

```
SET GLOBAL sql_mode='NO_BACKSLASH_ESCAPES';
```

This SQL mode also can be enabled automatically when the server starts by using the command-line option `--sql-mode=NO_BACKSLASH_ESCAPES` [394] or by setting `sql-mode=NO_BACKSLASH_ESCAPES` in the server option file (for example, `my.cnf` or `my.ini`, depending on your system). (Bug #8378, CVE-2006-2753)

References: See also Bug #8303.

- **Replication:** The dropping of a temporary table whose name contained a backtick (`` ` ``) character was not correctly written to the binary log, which also caused it not to be replicated correctly. (Bug #19188)
- Running `myisampack` followed by `myisamchk` with the `--unpack` [318] option would corrupt the `AUTO_INCREMENT` key. (Bug #12633)
- The patch for Bug #8303 broke the fix for Bug #8378 and was reverted.

  In string literals with an escape character (`\`) followed by a multi-byte character that had (`\`) as its second byte, the literal was not interpreted correctly. Now only next byte now is escaped, and not the entire multi-byte character. This means it is a strict reverse of the `mysql_real_escape_string()` function.
- RPM packages had spurious dependencies on Perl modules and other programs. (Bug #13634)
- The client libraries were not compiled for position-independent code on Solaris-SPARC and AMD x86_64 platforms. (Bug #18091, Bug #13159, Bug #14202)

C.1.7 Changes in MySQL 4.1.19 (2006-04-29)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This release includes the patches for recently reported security vulnerabilites in the MySQL client/server protocol. We would like to thank Stefano Di Paola `<stefano.dipaola@wisec.it>` for finding and reporting these to us.

Functionality Added or Changed

- **Security Enhancement:** Added the global `max_prepared_stmt_count` [420] system variable to limit the total number of prepared statements in the server. This limits the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. The current number of prepared statements is available through the `prepared_stmt_count` [423] system variable. (Bug #16365)

- **Packaging:** The `MySQL-shared-compat-4.1.X-.i386.rpm` shared compatibility RPMs no longer contain libraries for MySQL 5.0 and up. They now contain libraries for MySQL 3.23, 4.0, and 4.1.1 only. (Bug #19288)

- `InnoDB` now caches a list of unflushed files instead of scanning for unflushed files during a table flush operation. This improves performance when `--innodb_file_per_table` [1068] is set on a system with a large number of `InnoDB` tables. (Bug #15653)

- New `charset` command added to `mysql` command-line client. By typing `charset` *`name`* or `\C` *`name`* (such as `\C UTF8`), the client character set can be changed without reconnecting. (Bug #16217)

- When using the `GROUP_CONCAT()` [883] function where the `group_concat_max_len` [412] system variable was greater than 255, the result type differed depending on whether an `ORDER BY` clause was included: `BLOB` if it was, `VARBINARY` if it was not. (For nonbinary string arguments, the result was `TEXT` or `VARCHAR`.)

  Now an `ORDER BY` does not affect the result, which is `VARBINARY` (`VARCHAR`) only if `group_concat_max_len` [412] is less than or equal to 255, `BLOB` (`TEXT`) otherwise. (Bug #14169)

- Large file support was re-enabled for the MySQL server binary for the AIX 5.2 platform. (Bug #13571)

Bugs Fixed

- **Security Fix:** A malicious client, using specially crafted invalid login or `COM_TABLE_DUMP` packets was able to read uninitialized memory, which potentially, though unlikely in MySQL, could have led to an information disclosure. (,) Thanks to Stefano Di Paola `<stefano.dipaola@wisec.it>` for finding and reporting this bug. (CVE-2006-1516, CVE-2006-1517)

- **MySQL Cluster:** In some cases, `LOAD DATA INFILE` did not load all data into `NDB` tables. (Bug #17081)

- **MySQL Cluster:** The server would not compile with `NDB` support on AIX 5.2. (Bug #10776)

- **MySQL Cluster:** In a 2-node cluster with a node failure, restarting the node with a low value for `StartPartialTimeout` [1266] could cause the cluster to come up partitioned (“split-brain” issue).

  A similar issue could occur when the cluster was first started with a sufficiently low value for this parameter. (Bug #16447, Bug #18612)

- **MySQL Cluster:** A timeout in the handling of an `ABORT` condition with more that 32 operations could yield a node failure. (Bug #18414)
- **MySQL Cluster:** A simultaneous `DROP TABLE` and table update operation utilising a table scan could trigger a node failure. (Bug #18597)
- **MySQL Cluster:** When replacing a failed master node, the replacement node could cause the cluster to crash from a buffer overflow if it had an excessively large amount of data to write to the cluster log. (Bug #18118)
- **MySQL Cluster:** A `DELETE` with a join in the `WHERE` clause failed to retrieve any records if both tables in the join did not have a primary key. (Bug #17249)
- **MySQL Cluster:** The cluster created a crashed replica of a table having an ordered index—or when logging was not enabled, of a table having a table or unique index—leading to a crash of the cluster following 8 successive restarts. (Bug #18298)
- **MySQL Cluster:** The `REDO` log would become corrupted (and thus unreadable) in some circumstances, due to a failure in the query handler. (Bug #17295)
- **MySQL Cluster:** Inserting and deleting `BLOB` column values while a backup was in process could cause data nodes to shut down. (Bug #14028)
- **MySQL Cluster:** No error message was generated for setting `NoOfFragmentLogFiles` [1260] too low. (Bug #13966)
- **MySQL Cluster:** In event of a node failure during a rollback, a “false” lock could be established on the backup for that node, which lock could not be removed without restarting the node. (Bug #18352)
- **MySQL Cluster:** No error message was generated for setting `MaxNoOfAttributes` [1261] too low. (Bug #13965)
- **MySQL Cluster:** A node restart immediately following a `CREATE TABLE` would fail.

  Important

  This fix supports 2-node Clusters only.

  (Bug #18385)
- **MySQL Cluster:** Backups could fail for large clusters with many tables, where the number of tables approached `MaxNoOfTables` [1262]. (Bug #17607)
- **MySQL Cluster:** An `UPDATE` with an inner join failed to match any records if both tables in the join did not have a primary key. (Bug #17257)
- **MySQL Cluster:** Restarting nodes were permitted to start and join the cluster too early. (Bug #16772)
- **MySQL Cluster:** `ndb_delete_all` ran out of memory when processing tables containing `BLOB` columns. (Bug #16693)
- **MySQL Cluster:** On systems with multiple network interfaces, data nodes would get “stuck” in startup phase 2 if the interface connecting them to the management server was working on node startup while the interface interconnecting the data nodes experienced a temporary outage. (Bug #15695)
- **Replication:** Use of `TRUNCATE TABLE` for a `TEMPORARY` table on a master server was propagated to slaves properly, but slaves did not decrement the `Slave_open_temp_tables` [454] counter properly. (Bug #17137)

- The `IN`-to-`EXISTS` transformation was making a reference to a parse tree fragment that was left out of the parse tree. This caused problems with prepared statements. (Bug #18492)
- Conversion of a number to a `CHAR UNICODE` string returned an invalid result. (Bug #18691)
- The `mysql_close()` C API function leaked handles for shared-memory connections on Windows. (Bug #15846)
- `MyISAM`: Keys for which the first part of the key was a `CHAR` or `VARCHAR` column using the UTF-8 character set and longer than 254 bytes could become corrupted. (Bug #17705)
- A query using WHERE (*`column_1`*, *`column_2`*) IN ((*`value_1`*, *`value_2`*)[, (..., ...), ...]) would return incorrect results. (Bug #16248)
- The euro sign (€) was not stored correctly in columns using the `latin1_german1_ci` or `latin1_general_ci` collation. (Bug #18321)
- If `InnoDB` encountered a `HA_ERR_LOCK_TABLE_FULL` error and rolled back a transaction, the transaction was still written to the binary log. (Bug #18283)
- A `FULLTEXT` query in a `UNION` could result in unexpected behavior. (Bug #16893)
- A key on a `MEMORY` table would sometimes fail to match a row. (Bug #12796)
- When running a query that contained a `GROUP_CONCAT(SELECT GROUP_CONCAT(...) )` [883], the result was `NULL` except in the `ROLLUP` part of the result, if there was one. (Bug #15560)
- Connecting to a server with a UCS2 default character set with a client using a non-UCS2 character set crashed the server. (Bug #18004)
- Security Improvement: GRANTs to users with wildcards in their host information could be erroneously applied to similar users with the same user name and similar wildcards. For example, a privilege granted to `foo@%` are also applied to user `foo@192.%`. (Bug #14385)
- `LOAD DATA FROM MASTER` produced invalid warnings and `Packet out of order` errors when the database already existed on the slave. (Bug #15302)
- Dropping `InnoDB` constraints named *`tbl_name`*`_ibfk_0` could crash the server. (Bug #16387)
- A `LOCK TABLES` statement that failed could cause `MyISAM` not to update table statistics properly, causing a subsequent `CHECK TABLE` to report table corruption. (Bug #18544)
- `CAST(`*`double`*` AS SIGNED INT)` [859] for large *`double`* values outside the signed integer range truncated the result to be within range, but the result sometimes had the wrong sign, and no warning was generated. (Bug #15098)
- For single-`SELECT` union constructs of the form (SELECT ... ORDER BY *`order_list1`* [LIMIT *`n`*]) ORDER BY *`order_list2`*, the `ORDER BY` lists were concatenated and the `LIMIT` clause was ignored. (Bug #18767)
- Killing a long-running query containing a subquery could cause a server crash. (Bug #14851)
- Security improvement: In grant table comparisons, improper use of a `latin1` collation caused some host name matches to be true that should have been false. Thanks to Deomid Ryabkov for finding this bug and proposing a solution. (Bug #15756)
- Index corruption could occur in cases when `key_cache_block_size` [416] was not a multiple of the `myisam-block-size` [390] value (for example, with `--key_cache_block_size=1536` [416] and `--myisam-block-size=1024` [390]). (Bug #19079)

- `mysql_reconnect()` sent a `SET NAMES` statement to the server, even for pre-4.1 servers that do not understand the statement. (Bug #18830)
- A race condition could occur when dropping the adaptive hash index for a B-tree page in `InnoDB`. (Bug #16582)
- `SET` value definitions containing commas were not rejected. Now a definition such as `SET('a,b','c,d')` results in an error. (Bug #15316)
- The `-lmtmalloc` library was removed from the output of `mysql_config` on Solaris, as it caused problems when building `DBD::mysql` (and possibly other applications) on that platform that tried to use `dlopen()` to access the client library. (Bug #18322)
- Attempting to set the default value of an `ENUM` or `SET` column to `NULL` caused a server crash. (Bug #19145)
- The server was always built as though `--with-extra-charsets=complex` [97] had been specified. (Bug #12076)
- `UNCOMPRESS(NULL)` [869] could cause subsequent `UNCOMPRESS()` [869] calls to return `NULL` for legal non-`NULL` arguments. (Bug #18643)
- Setting the `myisam_repair_threads` [421] system variable to a value larger than 1 could cause corruption of large `MyISAM` tables. (Bug #11527)
- MySQL would not compile on Linux distributions that use the `tinfo` library. (Bug #18912)
- Avoid trying to include `<asm/atomic.h>` when it doesn't work in C++ code. (Bug #13621)
- Executing `SELECT` on a large table that had been compressed within `myisampack` could cause a crash. (Bug #17917)
- Binary distributions for Solaris contained files with group ownership set to the nonexisting `wheel` group. Now the `bin` group is used. (Bug #15562)
- IA-64 RPM packages for Red Hat and SuSE Linux that were built with the `icc` compiler incorrectly depended on `icc` runtime libraries. (Bug #16662)
- `SELECT ... WHERE column LIKE 'A%'`, when *`column`* had a key and used the `latin2_czech_cs` collation, caused the wrong number of rows to be returned. (Bug #17374)
- A call to `MIN()` [884] with a `CASE` [790] expression as its argument could return a nonminimum value. (Bug #17896)
- A `FULLTEXT` query in a prepared statement could result in unexpected behavior. (Bug #14496)
- `MYSQL_STMT` objects were not preserved following a connection reset. Attempting to operate on them afterward caused the server to crash. (Bug #12744)
- `SELECT COUNT(*)` for a `MyISAM` table could return different results depending on whether an index was used. (Bug #14980)
- Creating a table in an `InnoDB` database with a column name that matched the name of an internal `InnoDB` column (including `DB_ROW_ID`, `DB_TRX_ID`, `DB_ROLL_PTR` and `DB_MIX_ID`) would cause a crash. MySQL now returns Error 1005 `Cannot create table` with `errno` set to -1. (Bug #18934)
- `mysql_config` returned incorrect libraries on `x86_64` systems. (Bug #13158)
- Repeated invocation of `my_init()` and `my_end()` caused corruption of character set data and connection failure. (Bug #6536)

- `mysqldump` tried to dump data from a view. (In MySQL 4.1, this applies when connecting to a server from MySQL 5.0 or higher.) (Bug #16389)
- `MySQL-shared-compat-4.1.15-0.i386.rpm`, `MySQL-shared-compat-4.1.16-0.i386.rpm`, and `MySQL-shared-compat-4.1.18-0.i386.rpm` incorrectly depended on `glibc` 2.3 and could not be installed on a `glibc` 2.2 system. (Bug #16539)
- Index prefixes for `utf8 VARCHAR` columns did not work for `UPDATE` statements. (Bug #19080)
- Character set conversion of string constants for `UNION` of constant and table column was not done when it was safe to do so. (Bug #15949)
- During conversion from one character set to `ucs2`, multi-byte characters with no `ucs2` equivalent were converted to multiple characters, rather than to `0x003F QUESTION MARK`. (Bug #15375)

C.1.8 Changes in MySQL 4.1.18 (2006-01-27)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- **MySQL Cluster:** More descriptive warnings are now issued when inappropriate logging parameters are set in `config.ini`. (Formerly, the warning issued was simply `Could not add logfile destination`.) (Bug #11331)
- `libmysqlclient` now uses versioned symbols with GNU ld. (Bug #3074)

Bugs Fixed

- **Replication:** The `--replicate-do` and `--replicate-ignore` options were not being enforced on multiple-table statements. (Bug #16487, Bug #15699)
- A `CREATE TABLE ... SELECT ...` on an equation involving `DOUBLE` values could result in the table being created with columns too small to hold the equation result. (Bug #9855)
- A prepared statement created from a `SELECT ... LIKE` query (such as `PREPARE stmt1 FROM 'SELECT col_1 FROM tedd_test WHERE col_1 LIKE ?';`) would begin to produce erratic results after being executed repeatedly numerous (thousands) of times. (Bug #12734)
- `UPDATE` statement crashed multi-byte character set `FULLTEXT` index if update value was almost identical to initial value only differing in some spaces being changed to . (Bug #16489)
- Single table `UPDATE` statements without `ORDER BY` clauses which updated the same indexed column that was being filtered on were optimized with a full index scan instead of a more appropriate index range scan. (Bug #15935)
- RPM packages had an incorrect `zlib` dependency. (Bug #15223)
- Running out of diskspace in the location specified by the `tmpdir` [432] option resulted in incorrect error message. (Bug #14634)
- Test suite `func_math` test returned warnings when the server was not compiled with `InnoDB` support. (Bug #15429)
- The `MBROverlaps` GIS function returned incorrect results. (Bug #14320)

- `STR_TO_DATE(1,NULL)` [838] caused a server crash. (Bug #15828, CVE-2006-3081)
- The length of a `VARCHAR()` column that used the `utf8` character set would increase each time the table was re-created in a stored procedure or prepared statement, eventually causing the `CREATE TABLE` statement to fail. (Bug #13134)

C.1.9 Changes in MySQL 4.1.17 (Not released)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- Support files for compiling with Visual Studio 6 have been removed. (Bug #15094)
- In the `latin5_turkish_ci` collation, the order of the characters `A WITH CIRCUMFLEX`, `I WITH CIRCUMLEX`, and `U WITH CIRCUMFLEX` was changed. If you have used these characters in any indexed columns, you should rebuild those indexes. (Bug #13421)
- Internal `sha1_result` function renamed to `mysql_sha1_result` to prevent conflicts with other projects. (Bug #13944)

Bugs Fixed

- **MySQL Cluster:** A node which failed during cluster startup was sometimes not removed from the internal list of active nodes. (Bug #15587)
- **MySQL Cluster:** If an abort by the Transaction Coordinator timed out, the abort condition was incorrectly handled, causing the transaction record to be released prematurely. (Bug #15685)
- **MySQL Cluster:** Under some circumstances, it was possible for a restarting node to undergo a forced shutdown. (Bug #15632)
- **MySQL Cluster:** There was a small window for a node failure to occur during a backup without an error being reported. (Bug #15425)
- **MySQL Cluster:** A memory leak occurred when performing ordered index scans using indexes on columns larger than 32 bytes. This would eventually lead to the forced shutdown of all `mysqld` server processes used with the cluster. (Bug #13078)
- **Cluster API:** Upon the completion of a scan where a key request remained outstanding on the primary replica and a starting node died, the scan did not terminate. This caused incomplete error handling for the failed node. (Bug #15908)
- Multiple-table update operations were counting updates and not updated rows. As a result, if a row had several updates it was counted several times for the “rows matched” value but updated only once. (Bug #15028)
- `InnoDB`: Comparison of indexed `VARCHAR CHARACTER SET ucs2 COLLATE ucs2_bin` columns using `LIKE` could fail. (Bug #14583)
- Performing a `RENAME TABLE` on an `InnoDB` table when the server was started with the `--innodb_file_per_table` [1068] option and the data directory was a symlink caused a server crash. (Bug #15991)

- Characters in the gb2312 and euckr character sets which did not have Unicode mappings were truncated. (Bug #15377)
- Using an aggregate function as the argument for a HAVING clause resulted in the aggregate function always returning FALSE. (Bug #14274)
- SELECT queries that began with an opening parenthesis were not being placed in the query cache. (Bug #14652)
- DELETE could report full-text index corruption (Invalid key for table ...) if the index was built with repair-by-sort, the data in the full-text index used UCA collation, and some word appeared in the data terminated by a 0xC2A0 character as well as by other nonletter characters. (Bug #11336)
- InnoDB: If foreign_key_checks [411] was 0, InnoDB permitted inconsistent foreign keys to be created. (Bug #13778)
- CAST(... AS TIME) [859] operations returned different results when using versus not using prepared-statement protocol. (Bug #15805)
- The COALESCE() [784] function truncated data in a TINYTEXT column. (Bug #15581)
- BDB: A DELETE, INSERT, or UPDATE of a BDB table could cause the server to crash where the query contained a subquery using an index read. (Bug #15536)
- Symbolic links did not function properly on Windows platforms. (Bug #14960, Bug #14310)
- Certain CREATE TABLE ... AS ... statements involving ENUM columns could cause server crash. (Bug #12913)
- Using CAST() [859] to convert values with long fractional or exponent parts to TIME returned wrong results. (Bug #12440)
- A race condition when creating temporary files caused a deadlock on Windows with threads in Opening tables or Waiting for table states. (Bug #12071)
- Certain permission management statements could create a NULL host name for a user, resulting in a server crash. (Bug #15598)
- Issuing a DROP USER statement could cause some users to encounter a *hostname* is not permitted to connect to this MySQL server error. (Bug #15775)
- For InnoDB tables, using a column prefix for a utf8 column in a primary key caused Cannot find record errors when attempting to locate records. (Bug #14056)
- Access Denied error could be erroneously returned with specific grant combinations under high load. (Bug #7209)
- Piping the fill_help_tables.sql file into mysqld resulted in a syntax error. (Bug #15965)
- An INSERT ... SELECT statement between tables in a MERGE set can return errors when statement involves insert into child table from merge table or vice-versa. (Bug #5390)

C.1.10 Changes in MySQL 4.1.16 (2005-11-29)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- MySQL now supports character set conversion for seven additional `cp950` characters into the `big5` character set: `0xF9D6`, `0xF9D7`, `0xF9D8`, `0xF9D9`, `0xF9DA`, `0xF9DB`, and `0xF9DC`.

  Note

  If you move data containing these additional characters to an older MySQL installation which does not support them, you may encounter errors.

  (Bug #12476)

- When a date column is set `NOT NULL` and contains `0000-00-00`, it will be updated for UPDATE statements that contains *`columnname`* `IS NULL` in the WHERE clause. (Bug #14186)

- The `MySQL-server` RPM now explicitly assigns the `mysql` system user to the `mysql` user group during the postinstallation process. This corrects an issue with upgrading the server on some Linux distributions whereby a previously existing `mysql` user was not changed to the `mysql` group, resulting in wrong groups for files created following the installation. (Bug #12823)

- The `CHAR()` [794] function now takes an optional `USING` *`charset`* clause that may be used to produce a result in a specific character set rather than in the connection character set.

- When executing single-table `UPDATE` or `DELETE` queries containing an `ORDER BY ... LIMIT` *`N`* clause, but not having any `WHERE` clause, MySQL can now take advantage of an index to read the first *`N`* rows in the ordering specified in the query. If an index is used, only the first *`N`* records will be read, as opposed to scanning the entire table. (Bug #12915)

Bugs Fixed

- **MySQL Cluster:** Creating a table with packed keys failed silently. `NDB` now supports the `PACK_KEYS` option to `CREATE TABLE` correctly. (Bug #14514)

- **MySQL Cluster:** `REPLACE` failed when attempting to update a primary key value in a Cluster table. (Bug #14007)

- **MySQL Cluster:** Repeated transactions using unique index lookups could cause a memory leak leading to error 288, `Out of index operations in transaction coordinator`. (Bug #14199)

- **MySQL Cluster:** Placing multiple `[tcp default]` sections in the cluster's `config.ini` file crashed `ndb_mgmd`. (The process now exits gracefully in such cases, with an appropriate error message.) (Bug #13611)

- **MySQL Cluster:** The `perror` utility included with the `MySQL-Server` RPM did not provide support for the `--ndb` [360] option. It now supports this option, and so can be used to obtain error message text for MySQL Cluster error codes. (Bug #13740)

- **Replication:** On Windows, the server could crash during shutdown if both replication threads and normal client connection threads were active. (Bug #11796)

- **Replication:** Multiple update queries using any type of subquery would be ignored by a replication slave when a condition such as `--replicate-ignore-table` [1177] like condition was used. (Bug #13236)

- **Replication:** `InnoDB`: During replication, There was a failure to record events in the binary log that still occurred even in the event of a `ROLLBACK`. For example, this sequence of commands:

```
BEGIN;
```

```
CREATE TEMPORARY TABLE t1 (a INT) ENGINE=INNODB;
ROLLBACK;
INSERT INTO t1 VALUES (1);
```

would succeed on the replication master as expected. However, the `INSERT` would fail on the slave because the `ROLLBACK` would (erroneously) cause the `CREATE TEMPORARY TABLE` statement not to be written to the binlog. (Bug #7947)

- **Replication:** An `UPDATE` query using a join would be executed incorrectly on a replication slave. (Bug #12618)
- Given a column *`col_name`* defined as `NOT NULL`, a `SELECT ... FROM ... WHERE col_name IS NULL` query following `SHOW TABLE STATUS` would erroneously return a nonempty result. (Bug #13535)
- The default value of `query_prealloc_size` [425] was set to 8192, lower than its minimum of 16384. The minimum has been lowered to 8192. (Bug #13334)
- `make` failed when attempting to build MySQL in different directory other than that containing the source. (Bug #11827)
- `CREATE TABLE tbl_name (...) SELECT ...` could crash the server and write invalid data into the `.frm` file if the `CREATE TABLE` and `SELECT` both contained a column with the same name. Also, if a default value is specified in the column definition, it is now actually used. (Bug #14480)
- `InnoDB`: Pad UTF-8 `VARCHAR` columns with `0x20`. Pad UCS2 `CHAR` columns with `0x0020`. (Bug #10511)
- Queries of the form `(SELECT ...) ORDER BY ...` were being treated as a `UNION`. This improperly resulted in only distinct values being returned (because `UNION` by default eliminates duplicate results). Also, references to column aliases in `ORDER BY` clauses following parenthesized `SELECT` statements were not resolved properly. (Bug #7672)
- On Windows, the server was not ignoring hidden or system directories that Windows may have created in the data directory, and would treat them as available databases. (Bug #4375)
- An expression in an `ORDER BY` clause failed with `Unknown column 'col_name' in 'order clause'` if the expression referred to a column alias. (Bug #11694)
- `TIMEDIFF()` [840], `ADDTIME()` [828], and `STR_TO_DATE()` [838] were not reporting that they could return `NULL`, so functions that invoked them might misinterpret their results. (Bug #14009)
- With `--log-slave-updates` [1173] `Exec_master_log_pos` of SQL thread lagged IO (Bug #13023)
- `LIKE` operations did not work reliably for the `cp1250` character set. (Bug #13347)
- `mysqladmin` and `mysqldump` would hang on SCO OpenServer. (Bug #13238)
- For `MyISAM` tables, incorrect query results or incorrect updates could occur under these conditions: There is a multiple-column index that includes a `BLOB` column that is not the last column in the index, and the statement performs a lookup on the index using key column values that have `NULL` for the `BLOB` column and that provide values for all columns up to the `BLOB` column and at least the next column in the index. (Bug #13814)
- Closed a memory leak in the SSL code. (Bug #14780)
- `PURGE MASTER LOGS` statement that used subquery for date crashed server. (Bug #10308)
- Multiple race conditions existed in OpenSSL, particularly noticeable on Solaris. (Bug #9270)

- A `UNION` of `DECIMAL` columns could produce incorrect results. (Bug #14216)
- Use of `WITH ROLLUP PROCEDURE ANALYSE()` could hang the server. (Bug #14138)
- For a table that had been opened with `HANDLER OPEN`, issuing `OPTIMIZE TABLE`, `ALTER TABLE`, or `REPAIR TABLE` caused a server crash. (Bug #14397)
- `ALTER TABLE ... ENABLE INDEXES` treated `NULL` values as equal when collecting index statistics for `MyISAM` tables, resulting in different statistics from those generated by `ANALYZE TABLE` and causing the optimizer to make poor index choices later. The same problem occurred for bulk insert statistics collection. Now `NULL` values are treated as unequal, just as for `ANALYZE TABLE`. (Bug #9622)
- A `LIMIT`-related optimization failed to take into account that `MyISAM` table indexes can be disabled, causing Error 124 when it tried to use such an index. (Bug #14616)
- Corrected a memory-copying problem for `big5` values when using `icc` compiler on Linux IA-64 systems. (Bug #10836)
- `LOAD DATA INFILE` would not accept the same character for both the `ESCAPED BY` and the `ENCLOSED BY` clauses. (Bug #11203)
- An update of a `CSV` table could cause a server crash. (Bug #13894)
- Full-text indexing/searching failed for words that end with more than one apostrophe. (Bug #5686)
- Character set conversion was not being done for `FIND_IN_SET()` [796]. (Bug #13751)
- The endian byte in for spatial values in WKB format was not consistently respected. (Bug #12839)
- Creating a table containing an `ENUM` or `SET` column from within a stored procedure or prepared statement caused a server crash later when executing the procedure or statement. (Bug #14410)
- Use of *`col_name`* `= VALUES(`*`col_name`*`)` in the `ON DUPLICATE KEY UPDATE` clause of an `INSERT` statement failed with an `Column '`*`col_name`*`' in field list is ambiguous` error. (Bug #13392)
- `SELECT DISTINCT CHAR(`*`col_name`*`)` returned incorrect results after `SET NAMES utf8`. (Bug #13233)
- Maximum values were handled incorrectly for command-line options of type `GET_LL`. (Bug #12925)
- `CAST(1E+300 TO SIGNED INT)` [859] produced an incorrect result on little-endian machines. (Bug #13344)
- The server did not take character set into account in checking the width of the `mysql.user.Password` column. As a result, it could incorrectly generate long password hashes even if the column was not long enough to hold them. (Bug #13064)
- The `--interactive-timeout` and `--slave-net-timeout` options for `mysqld` were not being obeyed on Mac OS X and other BSD-based platforms. (Bug #8731)
- `mysqld_safe` did not correctly start the `-max` version of the server (if it was present) if the `--ledir` option was given. (Bug #13774)
- Issuing `STOP SLAVE` after having acquired a global read lock with `FLUSH TABLES WITH READ LOCK` caused a deadlock. Now `STOP SLAVE` is generates an error in such circumstances. (Bug #10942)
- Deletes from a `CSV` table could cause table corruption. (Bug #14672)
- Selecting from a table in both an outer query and a subquery could cause a server crash. (Bug #14482)

- Character set file parsing during `mysql_real_connect()` read past the end of a memory buffer. (Bug #6413)
- Specifying `--default-character-set=cp-932` for `mysqld` would cause SQL scripts containing comments written using that character set to fail with a syntax error. (Bug #13487)
- On BSD systems, the system `crypt()` call could return an error for some salt values. The error was not handled, resulting in a server crash. (Bug #13619)
- Statements of the form `CREATE TABLE ... SELECT ...` that created a column with a multi-byte character set could incorrectly calculate the maximum length of the column, resulting in a `Specified key was too long` error. (Bug #14139)
- The example configuration files supplied with MySQL distributions listed the `thread_cache_size` [431] variable as `thread_cache`. (Bug #13811)
- Perform character set conversion of constant values whenever possible without data loss. (Bug #10446)
- Portability fixes to support OpenSSL 0.9.8a. (Bug #14221)
- Non-`latin1` object names were written with wrong character set to grant tables. (Bug #14406)
- `PROCEDURE ANALYSE()` could suggest a data type with a negative display width. (Bug #10716)
- `mysql_fix_privilege_tables.sql` contained an erroneous comment that resulted in an error when the file contents were processed. (Bug #14469)
- When the `DATE_FORMAT()` [832] function appeared in both the `SELECT` and `ORDER BY` clauses of a query but with arguments that differ by case (for example, `%m` and `%M`), incorrect sorting may have occurred. (Bug #14016)

C.1.11 Changes in MySQL 4.1.15 (2005-10-13)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- **MySQL Cluster:** The parsing of the `CLUSTERLOG` command by `ndb_mgm` was corrected to permit multiple items. (Bug #12833)
- **Replication:** Better detection of connection timeout for replication servers on Windows enables elimination of extraneous `Lost connection` errors in the error log. (Bug #5588)
- A new command line argument was added to `mysqld` to ignore client character set information sent during handshake, and use server side settings instead, to reproduce 4.0 behavior :

  ```
  mysqld --skip-character-set-client-handshake
  ```

  (Bug #9948)
- When using `IF NOT EXISTS` with `CREATE DATABASE` or `CREATE TABLE`, a warning now is generated if the database or table already exists. : (Bug #6008)
- Added the `myisam_stats_method` [422], which controls whether `NULL` values in indexes are considered the same or different when collecting statistics for `MyISAM` tables. This influences the query optimizer as described in Section 7.4.4, “`MyISAM` Index Statistics Collection”. (Bug #12232)

- The limit of 255 characters on the input buffer for `mysql` on Windows has been lifted. The exact limit depends on what the system permits, but can be up to 64KB characters. A typical limit is 16KB characters. (Bug #12929)
- `RAND()` [822] no longer permits nonconstant initializers. (Previously, the effect of nonconstant initializers is undefined.) (Bug #6172)

Bugs Fixed

- **MySQL Cluster:** With two `mgmd` processes in a cluster, `ndb_mgm` output for `SHOW` would display the same IP address for both processes, even when they were on different hosts. (Bug #11595)
- **MySQL Cluster:** Adding an index to a table with a large number of columns (more then 100) crashed the storage node. (Bug #13316)
- **MySQL Cluster:** Improved error messages related to file system issues. (Bug #11218)
- **MySQL Cluster:** The cluster management client `START BACKUP` command could be interrupted by a `SHOW` command. (Bug #13054)
- **MySQL Cluster:** Multiple `ndb_mgmd` processes in a cluster did not know each other's IP addresses. (Bug #12037)
- **MySQL Cluster:** When it could not copy a fragment, `ndbd` exited without printing a message about the condition to the error log. Now the message is written. (Bug #12900)
- **MySQL Cluster:** When a schema was detected to be corrupt, `ndb` neglected to close it, resulting in a `file already open` error if the schema was opened again later. written. (Bug #12027)
- **MySQL Cluster:** `LOAD DATA INFILE` with a large data file failed. (Bug #10694)
- **MySQL Cluster:** A cluster shutdown following the crash of a data node failed to terminate any remaining node processes, even though `ndb_mgm` showed the shutdown request as having been completed. (Bug #9996, Bug #10938, Bug #11623)
- **MySQL Cluster:** When deleting a great many (tens of thousands of) rows at once from an `NDB` table, an improperly dereferenced pointer could cause the `mysqld` process to crash. (Bug #9282)
- **MySQL Cluster:** Invalid values in `config.ini` caused `ndb_mgmd` to crash. (Bug #12043)
- **MySQL Cluster:** An `ALTER TABLE` statement caused loss of data stored prior to the issuing of the command. (Bug #12118)
- **MySQL Cluster:** Updating a column of one of the `TEXT` types during a cluster backup could cause the `ndbd` process to crash, due to the incorrect use of charset-normalized reads. This could also lead to character data having the wrong lettercase in the backup if such a column was updated during the backup; for example, supposing that the column used `latin_ci`, then “`aAa`” might be stored in the backup as “`AAA`”. (Bug #12950)
- **MySQL Cluster:** MySQL failed to compile when `--with-ndb-ccflags` was specified. (Bug #11538)
- **MySQL Cluster:** When a `Disk is full` condition occurred, `ndbd` exited without reporting this condition in the error log. (Bug #12716)
- **Replication:** The `--replicate-rewrite-db` [1177] and `--replicate-do-table` [1177] options did not work for statements in which tables were aliased to names other than those listed by the options. (Bug #11139)
- **Replication:** If a `DROP DATABASE` fails on a master server due to the presence of a nondatabase file in the database directory, the master have the database tables deleted, but not the slaves. To deal with

failed database drops, we now write `DROP TABLE` statements to the binary log for the tables so that they are dropped on slaves. (Bug #4680)

- **Replication:** When any `--replicate-wild-* option` is used, the slave ignores `SET ONE_SHOT TIME_ZONE` statements as belonging to a nonreplicated table. (Bug #12542)
- `SHOW CREATE TABLE` did not display any `FOREIGN KEY` clauses if a temporary file could not be created. Now `SHOW CREATE TABLE` displays an error message in an SQL comment if this occurs. (Bug #13002)
- The counters for the `Key_read_requests` [452], `Key_reads` [452], `Key_write_requests` [452], and `Key_writes` [452] status variables were changed from `unsigned long` to `unsigned longlong` to accommodate larger values before the variables roll over and restart from 0. (Bug #12920)
- A `SELECT DISTINCT` query with a constant value for one of the columns would return only a single row. (Bug #12625)
- A client connection thread cleanup problem caused the server to crash when closing the connection if the binary log was enabled. (Bug #12517)
- The `ARCHIVE` storage engine does not support deletes, but it was possible to delete by using `DELETE` or `TRUNCATE TABLE` with a `FEDERATED` table that points to an `ARCHIVE` table. (Bug #12836)
- If a client has opened an `InnoDB` table for which the `.ibd` file is missing, `InnoDB` would not honor a `DROP TABLE` statement for the table. (Bug #12852)
- `UNION` of two `DECIMAL` columns returned the wrong field type. (Bug #13372)
- If special characters such as `'_'`, `'%'`, or the escape character were included within the prefix of a column index, `LIKE` pattern matching on the indexed column did not return the correct result. (Bug #13046, Bug #13919)
- For `VARCHAR` columns with the `ucs2` character set, `InnoDB` trimmed trailing `0x20` bytes rather than `0x0020` words, resulting in incorrect index lookups later. (Bug #12178)
- Display of the `AUTO_INCREMENT` attribute by `SHOW CREATE TABLE` was not controlled by the `NO_FIELD_OPTIONS` [459] SQL mode as it should have been. (Bug #7977)
- The `CHECKSUM TABLE` statement returned incorrect results for tables with deleted rows. After upgrading, users who used stored checksum information to detect table changes should rebuild their checksum data. (Bug #12296)
- On Windows, the server was preventing tables from being created if the table name was a prefix of a forbidden name. For example, `nul` is a forbidden name because it is the same as a Windows device name, but a table with the name of `n` or `nu` was being forbidden as well. (Bug #12325)
- Deadlock occurred when several account management statements were run (particularly between `FLUSH PRIVILEGES`/`SET PASSWORD` and `GRANT`/`REVOKE` statements). (Bug #12423)
- Aggregate functions sometimes incorrectly were permitted in the `WHERE` clause of `UPDATE` and `DELETE` statements. (Bug #13180)
- The server could over-allocate memory when performing a `FULLTEXT` search for stopwords only. (Bug #13582)
- Reverted a change introduced in MySQL 4.1.13 (`SHOW FIELDS` truncated the `TYPE` column to 40 characters). This fix was reverted for MySQL 4.1 because it broke existing applications. The fix will be made in MySQL 5.0 instead (5.0.13). (Bug #12817)

References: The patch for the following bug was reverted: Bug #7142.

- `SELECT GROUP_CONCAT(...) FROM DUAL` in a subquery could cause the client to hang. (Bug #12861)
- A concurrency problem for `CREATE ... SELECT` could cause a server crash. (Bug #12845)
- `CHECKSUM TABLE` locked `InnoDB` tables and did not use a consistent read. (Bug #12669)
- `DELETE` or `UPDATE` for an indexed `MyISAM` table could fail. This was due to a change in end-space comparison behavior from 4.0 to 4.1. (Bug #12565)
- `MEMORY` tables using `B-Tree` index on 64-bit platforms could produce false table is full errors. (Bug #12460)
- A prepared statement failed with `Illegal mix of collations` if the client character set was `utf8` and the statement used a table that had a character set of `latin1`. (Bug #12371)
- Performing an `IS NULL` [783] check on the `MIN()` [884] or `MAX()` [884] of an indexed column in a complex query could produce incorrect results. (Bug #12695)
- On Windows when the `--innodb_buffer_pool_awe_mem_mb` [1067] option has been given, the server detects whether AWE support is available and has been compiled into the server, and displays an appropriate error message if not. (Bug #6581)
- `InnoDB` was too permissive with `LOCK TABLE ... READ LOCAL` and permitted new inserts into the table. Now `READ LOCAL` is equivalent to `READ` for `InnoDB`. This will cause slightly more locking in `mysqldump`, but makes `InnoDB` table dumps consistent with `MyISAM` table dumps. (Bug #12410)
- For queries with `DISTINCT` and `WITH ROLLUP`, the `DISTINCT` should be applied after the rollup operation, but was not always. (Bug #12887)
- `ALTER TABLE db_name.t RENAME t` did not move the table to default database unless the new name was qualified with the database name. (Bug #11493)
- MySQL would pass an incorrect key length to storage engines for `MIN()` [884]. This could cause spurious warnings such as `InnoDB: Warning: using a partial-field key prefix in search` to appear in the `.err` log. (Bug #13218, Bug #11039)
- The data type for `DECIMAL` columns was not respected when updating the column from another column. For example, updating a `DECIMAL(10,1)` column with the value from a `DECIMAL(10,5)` column resulted in a `DECIMAL(10,5)` value being stored. Similarly, altering a column with a `DECIMAL(10,5)` data type to a `DECIMAL(10,1)` data type did not properly convert data values. (Bug #7598)
- Shared-memory connections were not working on Windows. (Bug #12723)
- `LOAD DATA INFILE` did not respect the `NO_AUTO_VALUE_ON_ZERO` [458] SQL mode setting. (Bug #12053)
- After changing the character set with `SET CHARACTER SET`, the result of the `GROUP_CONCAT()` [883] function was not converted to the proper character set. (Bug #12829)
- Queries against a `MERGE` table that has a composite index could produce incorrect results. (Bug #9112)
- `GROUP_CONCAT()` [883] ignored an empty string if it was the first value to occur in the result. (Bug #12863)
- `TRUNCATE TABLE` did not work with `TEMPORARY InnoDB` tables. (Bug #11816)

- An optimizer estimate of zero rows for a nonempty `InnoDB` table used in a left or right join could cause incomplete rollback for the table. (Bug #12779)
- Use of a user-defined function within the `HAVING` clause of a query resulted in an `Unknown column` error. (Bug #11553)
- Users created using an IP address or other alias rather than a host name listed in `/etc/hosts` could not set their own passwords. (Bug #12302)
- The value of `character_set_results` [408] could be set to `NULL`, but returned the string `"NULL"` when retrieved. (Bug #12363)
- Outer join elimination was erroneously applied for some queries that used a `NOT BETWEEN` condition, an `IN(value_list)` [784] condition, or an `IF()` [790] condition. (Bug #12102, Bug #12101)
- A `UNION` of long `utf8` `VARCHAR` columns was sometimes returned as a column with a `LONGTEXT` data type rather than `VARCHAR`. This could prevent such queries from working at all if selected into a `MEMORY` table because the `MEMORY` storage engine does not support the `TEXT` data types. (Bug #12537)
- A column that can be `NULL` was not handled properly for `WITH ROLLUP` in a subquery or view. (Bug #12885)
- Spatial index corruption could occur during updates. (Bug #9645)
- Queries that created implicit temporary tables could return incorrect data types for some columns. (Bug #11718)
- On HP-UX 11.x (PA-RISC), the `-L` option caused `mysqlimport` to crash. (Bug #12958)
- The `have_innodb` [413] read-only system variable could not be selected with `SELECT @@have_innodb`. (Bug #9613)
- After running `configure` with the `--with-embedded-privilege-control` option, the embedded server failed to build. (Bug #13501)
- The server crashed when one thread resized the query cache while another thread was using it. (Bug #12848)
- `mysqld_multi` now quotes arguments on command lines that it constructs to avoid problems with arguments that contain shell metacharacters. (Bug #11280)
- Comparisons involving row constructors containing constants could cause a server crash. (Bug #13356)
- `myisampack` did not properly pack `BLOB` values larger than 2^{24} bytes. (Bug #4214)
- The `LIKE ... ESCAPE` syntax produced invalid results when escape character was larger than one byte. (Bug #12611)
- MySQL programs in binary distributions for Solaris 8/9/10 x86 systems would not run on Pentium III machines. (Bug #6772)
- `MIN()` [884] and `MAX()` [884] sometimes returned a non-`NULL` value for an empty row set (for example, `SELECT MAX(1) FROM empty_table`). (Bug #12882)

C.1.12 Changes in MySQL 4.1.14 (2005-08-17)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please

consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- **MySQL Cluster:** Improved handling of the configuration variables `NoOfPagesToDiskDuringRestartACC`, `NoOfPagesToDiskAfterRestartACC`, `NoOfPagesToDiskDuringRestartTUP`, and `NoOfPagesToDiskAfterRestartTUP` should result in noticeably faster startup times for MySQL Cluster. (Bug #12149)
- **MySQL Cluster:** A new `-P` option is available for use with the `ndb_mgmd` client. When called with this option, `ndb_mgmd` prints all configuration data to `stdout`, then exits.
- The MySQL server now starts correctly with all combinations of `--basedir` [384] and `--datadir` [385], resolving an issue introduced by the original fix for this bug in MySQL 4.1.9. (Bug #7249)

  References: See also Bug #7518.
- Added support of where clause for queries with `FROM DUAL`. (Bug #11745)
- `SHOW CHARACTER SET` and `INFORMATION_SCHEMA` now properly report the `Latin1` character set as `cp1252`. (Bug #11216)
- If a thread (connection) has tables locked, the query cache is switched off for that thread. This prevents invalid results where the locking thread inserts values between a second thread connecting and selecting from the table. (Bug #12385)
- Added an optimization that avoids key access with `NULL` keys for the `ref` [567] method when used in outer joins. (Bug #12144)
- Added new query cache test for the embedded server to the test suite, there are now specific tests for the embedded and nonembedded servers. (Bug #9508)

Bugs Fixed

- **MySQL Cluster:** `NDB` ignored the `Hostname` option in the `[ndbd default]` section of the cluster configuration file. (Bug #12028)
- **MySQL Cluster:** `ndb_mgmd` leaked file descriptors. (Bug #11898)
- **MySQL Cluster:** The temporary tables created by an `ALTER TABLE` on an `NDB` table were visible to all SQL nodes in the cluster. (Bug #12055)
- **MySQL Cluster:** The output of `perror --help` did not display any information about the `--ndb` [360] option. (Bug #11999)
- **MySQL Cluster:** Attempting to create or drop tables during a backup would cause the cluster to shut down. (Bug #11942)
- **Replication:** Slave I/O threads were considered to be in the running state when launched (rather than after successfully connecting to the master server), resulting in incorrect `SHOW SLAVE STATUS` output. (Bug #10780)
- `SELECT @@local...` returned `@@session...` in the column header. (Bug #10724)
- The value of `max_connections_per_hour` was capped by the unrelated `max_user_connections` setting. (Bug #9947)

- Performing `DATE(LEFT(column,8))` [829] on a `DATE` column produces incorrect results. (Bug #12266)
- Renamed the `rest()` macro in `my_list.h` to `list_rest()` to avoid name clashes with user code. (Bug #12327)
- For prepared statements, the SQL parser did not disallow `?` parameter markers immediately adjacent to other tokens, which could result in malformed statements in the binary log. (For example, `SELECT * FROM t WHERE? = 1` could become `SELECT * FROM t WHERE0 = 1`.) (Bug #11299)
- Some subqueries of the form `SELECT ... WHERE ROW(...) IN (subquery)` were being handled incorrectly. (Bug #11867)
- References to system variables in an SQL statement prepared with `PREPARE` were evaluated during `EXECUTE` to their values at prepare time, not to their values at execution time. (Bug #9359)
- When two threads competed for the same table, a deadlock could occur if one thread also had a lock on another table through `LOCK TABLES` and the thread was attempting to remove the table in some manner while the other thread tried to place locks on both tables. (Bug #10600)
- A `UNION` query with `FULLTEXT` could cause server crash. (Bug #11869)
- `ISO-8601` formatted dates were not being parsed correctly. (Bug #7308)
- Character data truncated when GBK characters `0xA3A0` and `0xA1` are present. (Bug #11987)
- Two threads could potentially initialize different characters sets and overwrite each other. (Bug #12109)
- Comparisons like `SELECT "A\\" LIKE "A\\";` fail when using `SET NAMES utf8;`. (Bug #11754)
- Attempting to repair a table having a full-text index on a column containing words whose length exceeded 21 characters and where `myisam_repair_threads` [421] was greater than 1 would crash the server. (Bug #11684)
- `InnoDB`: Do not flush after each write, not even before setting up the doublewrite buffer. Flushing can be extremely slow on some systems. (Bug #12125)
- `SHOW BINARY LOGS` displayed a file size of 0 for all log files but the current one if the files were not located in the data directory. (Bug #12004)
- Mishandling of comparison for rows containing `NULL` values against rows produced by an `IN` subquery could cause a server crash. (Bug #12392)
- Concatenating `USER()` [876] or `DATABASE()` [872] with a column produced invalid results. (Bug #12351)
- Creation of the `mysql` group account failed during the RPM installation. (Bug #12348)
- `myisam.test` failed when server compiled using `--without-geometry` option. (Bug #11083)
- Pathame values for options such as `--basedir` [384] or `--datadir` [385] didn't work on Japanese Windows machines for directory names containing multi-byte characters having a second byte of `0x5C` (“`\`”). (Bug #5439)
- `myisampack` failed to delete `.TMD` temporary files when run with the `-T` option. (Bug #12235)
- `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` could fail with an erroneous “Column '*col_name*' specified twice” error. (Bug #10109)

- Multiplying `ABS()` [817] output by a negative number would return incorrect results. (Bug #11402)
- `big5` strings were not being stored in `FULLTEXT` index. (Bug #12075)
- `FLUSH TABLES WITH READ LOCK` combined with `LOCK TABLE .. WRITE` caused deadlock. (Bug #9459)
- `GROUP_CONCAT()` [883] sometimes returned a result with a different collation from that of its arguments. (Bug #10201)
- Incorrect error message displayed if user attempted to create a table in a nonexisting database using `CREATE` *`database_name.table_name`* syntax. (Bug #10407)
- The `LPAD()` [798] and `RPAD()` [800] functions returned the wrong length to `mysql_fetch_fields()`. (Bug #11311)
- The `mysql_info()` C API function could return incorrect data when executed as part of a multi-statement that included a mix of statements that do and do not return information. (Bug #11688)
- Queries with subqueries that contain outer joins could return wrong results. (Bug #11479)
- Corrected a problem with the optimizer incorrectly adding `NOT NULL` constraints, producing in incorrect results for complex queries. (Bug #11482)
- Creating a table with a `SET` or `ENUM` column with the `DEFAULT 0` clause caused a server crash if the table's character set was `utf8`. (Bug #11819)
- In SQL prepared statements, comparisons could fail for values not equally space-padded. For example, `SELECT 'a' = 'a ';` returns 1, but `PREPARE s FROM 'SELECT ?=?'; SET @a = 'a', @b = 'a '; PREPARE s FROM 'SELECT ?=?'; EXECUTE s USING @a, @b;` incorrectly returned 0. (Bug #9379)
- Updated dependency list for RPM builds to include missing dependencies such as `useradd` and `groupadd`. (Bug #12233)
- `mysql_fetch_fields()` returned incorrect length information for `MEDIUM` and `LONG TEXT` and `BLOB` columns. (Bug #9735)
- Corrected an optimizer problem with `NOT NULL` constraints within a subquery in an `UPDATE` statement that resulted in a server crash. (Bug #11868)
- For DMG installs on Mac OS X, the preinstallation and postinstallation scripts were being run only for new installations and not for upgrade installations, resulting in an incomplete installation process. (Bug #11380)
- `mysql_next_result()` returns incorrect value if final query in a batch fails. (Bug #12001)
- Prepared statement parameters could cause errors in the binary log if the character set was `cp932`. (Bug #11338)
- LIKE pattern matching using prefix index didn't return correct result. (Bug #11650)
- Multiple-table `UPDATE` queries using `CONVERT_TZ()` [828] would fail with an error. (Bug #9979)
- `GROUP_CONCAT` ignored the `DISTINCT` modifier when used in a query joining multiple tables where one of the tables had a single row. (Bug #12095)
- The C API function `mysql_stmt_reset()` did not clear error information. (Bug #11183)

- User variables were not automatically cast for comparisons, causing queries to fail if the column and connection character sets differed. Now when mixing strings with different character sets but the same coercibility, permit conversion if one character set is a superset of the other. (Bug #10892)
- Server-side prepared statements failed for columns with a character set of `ucs2`. (Bug #9442)

C.1.13 Changes in MySQL 4.1.13 (2005-07-15)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- **Security Fix:** A UDF library-loading vulnerability could result in a buffer overflow and code execution. (CVE-2005-2558)
- **Incompatible Change:** Previously, conversion of `DATETIME` values to numeric form by adding zero produced a result in `YYYYMMDDHHMMSS` format. The result of `DATETIME+0` is now in `YYYYMMDDHHMMSS.000000` format. (Bug #12268)
- **Replication:** Some data definition statements (`CREATE TABLE` where the table was not a temporary table, `TRUNCATE TABLE`, `DROP DATABASE`, and `CREATE DATABASE`) were not being written to the binary log after a `ROLLBACK`. This also caused problems with replication.

  Important

  As a result of this fix, the following statements now cause an implicit commit:

  - `CREATE TABLE`
  - `TRUNCATE TABLE`
  - `DROP DATABASE`
  - `CREATE DATABASE`

  (Bug #6883)
- System variables are now treated as having `SYSVAR` (system constant) coercibility. For example, `@@version` is now treated like `VERSION()` [876] and `@@character_set_client` is now treated like `CHARSET( USER() )` [870]. See Section 9.1.7.5, “Collation of Expressions”. (Bug #10904)
- `InnoDB`: When creating or extending an `InnoDB` data file, allocate at most one megabyte at a time for initializing the file. Previously, `InnoDB` used to allocate and initialize 1 or 8 megabytes of memory, even if a few 16-kilobyte pages were to be written. This fix improves the performance of `CREATE TABLE` in `innodb_file_per_table` [1068] mode.
- Added the `--add-drop-database` [292] option to `mysqldump`. (Bug #3716)
- Added `mysql_set_character_set()` C API function for setting the default character set of the current connection. This enables clients to affect the character set used by `mysql_real_escape_string()`. (Bug #8317)
- `SHOW BINARY LOGS` now displays a `File_size` column that indicates the size of each file.
- You can again refer to other tables in the `ON DUPLICATE KEY UPDATE` part of an `INSERT ... SELECT` statement as long as there is no `GROUP BY` in the `SELECT` part. One side effect of this is that

you may have to qualify nonunique column names in the values part of `ON DUPLICATE KEY UPDATE`. (Bug #9728, Bug #8147)

- Added the `--log-slow-admin-statements` [388] server option to request logging of slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` to the slow query log. These statements were logged in MySQL 4.0, but not in 4.1. (Bug #9141)
- `MEMORY` tables now support indexes of up to 500 bytes. See Section 13.4, "The `MEMORY` (`HEAP`) Storage Engine". (Bug #10566)
- The `table`, `type`, and `rows` columns of `EXPLAIN` output can now be `NULL`. This is required for using `EXPLAIN` on `SELECT` queries that use no tables, such as `EXPLAIN SELECT 1`). (Bug #9899)
- Expanded on information provided in general log and slow query log for prepared statements. (Bug #8367, Bug #9334)

Bugs Fixed

- **Security Fix:** A vulnerability in `zlib` could result in a buffer overflow and arbitrary code execution. Shortly after MySQL 4.1.13 was released, a second potential `zlib` security flaw was discovered and fixed. A patch for this flaw was applied to the 4.1.13 sources, and the result published as MySQL 4.1.13a. The affected binaries were rebuilt. (Bug #11844, CVE-2005-2096, CVE-2005-1849)
- **Security Fix:** On Windows systems, a user with any of the following privileges on `*.*` could crash `mysqld` by issuing a `USE LPT1;` or `USE PRN;` command:
  - `REFERENCES` [492]
  - `CREATE TEMPORARY TABLES` [491]
  - `GRANT OPTION` [491]
  - `CREATE` [491]
  - `SELECT`

  In addition, any of the commands `USE NUL;`, `USE CON;`, `USE COM1;`, or `USE AUX;` would report success even though the database was not in fact changed. (Bug #9148)
- **MySQL Cluster:** When trying to open a table that could not be discovered or unpacked, the cluster returned error codes which the MySQL server falsely interpreted as operating system errors. (Bug #10365)
- **MySQL Cluster:** `NDB` failed to build with `gcc` 4.0. (Bug #11377)
- **Replication:** `LOAD DATA ... REPLACE INTO ...` on a replication slave failed for an `InnoDB` table having a unique index in addition to the primary key. (Bug #11401)
- **Replication:** An `UPDATE` query containing a subquery caused replication to fail. (Bug #9361)
- **Replication:** An invalid comparison caused warnings for packet length in replication on 64-bit compilers. (Bug #11064)
- **Replication:** Queries of the form `UPDATE ... (SELECT ... ) SET ...` run on a replication master would crash all the slaves. (Bug #10442, CVE-2004-4380)
- Some internal functions did not take into account that, for multi-byte character sets, `CHAR` and `VARCHAR` columns could exceed 255 bytes, which could cause the server to crash. (Bug #11167)

- Queries with subqueries in the `FROM` clause were not being added to the query cache. (Bug #11522)
- Invoking the `DES_ENCRYPT()` [866] function could cause a server crash if the server was started without the `--des-key-file` [386] option. (Bug #11643)
- Incorrect results when searching using `IN()` [784] where search items included `NULL` and `0`. (Bug #9393)
- Queries with `ROLLUP` returned wrong results for expressions containing`GROUP BY` columns. (Bug #7894)
- `SHOW WARNINGS` with a `LIMIT 0` clause returned all messages rather than an empty result set. (Bug #11095)
- Using `#pragma interface` or `#pragma implementation` in source files caused portability issues for `cygwin`. (Bug #10241)
- Table names were not handled correctly when `lower_case_table_names = 2` [418] if the table name lettercase differed in the `FROM` and `WHERE` clauses. (Bug #9500)
- On Mac OS X, `libmysqlclient_r.a` now is built with `--fno-common` to make it possible to link a shared two-level namespace library against `libmysqlclient_r.a`. (Bug #10638)
- Optimizer performed range check when comparing unsigned integers to negative constants, could cause errors. (Bug #11185)
- The host name cache was not working. (Bug #10931)
- When used within a subquery, `SUBSTRING()` [802] returned an empty string. (Bug #10269)
- Possible `NULL` values in `BLOB` columns could crash the server when a `BLOB` was used in a `GROUP BY` query. (Bug #11295)
- A simultaneous `CREATE TABLE ... SELECT FROM` *`table`* and `ALTER TABLE` *`table`* on the same table caused the server to crash. (Bug #10224)
- `SHOW FIELDS` truncated the `TYPE` column to 40 characters.

  Note

  This fix was reverted in MySQL 4.1.15 because it broke existing applications.

  (Bug #7142)

  References: See also Bug #12817.
- The `LAST_DAY()` [836] failed to return `NULL` when supplied with an invalid argument. See Section 11.7, “Date and Time Functions”. (Bug #10568)
- Modifying a `CHAR` column with the `utf8` character set to a shorter length did not properly truncate values due to not computing their length in `utf8` character units. (Bug #11591)
- `mysqldump` could crash for illegal or nonexistent table names. (Bug #9358)
- Inserting a `DOUBLE` value into a `utf8` string column crashed the server on Windows. (Bug #10714)
- Corrected an optimization failure where a query returned an incorrect result for use of a newly populated table until the table was flushed. (Bug #11700)

- `mysqldump` crashed using the `--complete-insert` [293] option while dumping tables with a large number of long column names. (Bug #10286)
- The `mysql_config` script did not handle symbolic linking properly. (Bug #10986)
- `CASE` [790] operator returns incorrect result when its arguments are not constants and its return value is put into a regular or temporary table (temporary == created by SQL engine for `UNION`/nonindexed `GROUP BY` and such operations). (Bug #10151)
- For a `MERGE` table with `MyISAM` tables in other, symlinked, databases, `SHOW CREATE TABLE` reported the `MyISAM` tables using the name of the symlinked directory rather than the database name. (Bug #8183)
- `INSERT ... ON DUPLICATE KEY UPDATE` with `MERGE` tables, which do not have unique indexes, caused the server to crash. (Bug #10400)
- A three byte buffer overflow in the client functions caused improper exiting of the client when reading a command from the user. (Bug #10841)
- `mysqld_safe` would sometimes fail to remove the pid file for the old `mysql` process after a crash. As a result, the server would fail to start due to a false `A mysqld process already exists...` error. (Bug #11122)
- Selecting the result of an aggregate function for an `ENUM` or `SET` column within a subquery could result in a server crash. (Bug #11821)
- The server timed out SSL connections too quickly on Windows. (Bug #8572)
- `mysqldump --xml` did not format `NULL` column values correctly. (Bug #9657)
- When used in joins, `SUBSTRING()` [802] failed to truncate to zero those string values that could not be converted to numbers. (Bug #10124)
- `DES_ENCRYPT()` [866] and `DES_DECRYPT()` [866] require SSL support to be enabled, but were not checking for it. Checking for incorrect arguments or resource exhaustion was also improved for these functions. (Bug #10589)
- For a `UNION` that involved long string values, values were not being converted correctly to `TEXT` values. (Bug #10025)
- The incorrect sequence of statements `HANDLER tbl_name READ index_name NEXT` without a preceding `HANDLER tbl_name READ index_name = (value_list)` for an `InnoDB` table resulted in a server crash rather than an error. (Bug #5373)
- A `CREATE TABLE db_name.tbl_name LIKE ...` statement would crash the server when no database was selected. (Bug #11028)
- IP addresses not shown in `ndb_mgm SHOW` command on second ndb_mgmd (or on ndb_mgmd restart). (Bug #11596)
- MySQL sometimes reported erroneously that certain character values had crashed a table when trying to convert other character sets to UTF-8. (Bug #9557)
- Setting `@@sql_mode = NULL` caused an erroneous error message. (Bug #10732)
- `ALTER TABLE ... ENABLE INDEXES` treated `NULL` values as equal when collecting index statistics for `MyISAM` tables, resulting in different statistics from those generated by `ANALYZE TABLE` and causing

the optimizer to make poor index choices later. The same problem occurred for bulk insert statistics collection. Now `NULL` values are treated as unequal, just as for `ANALYZE TABLE`. (Bug #9622)

- `CREATE TABLE t AS SELECT UUID()` created a `VARCHAR(12)` column, which is too small to hold the 36-character result from `UUID()` [879]. (Bug #9535)
- A problem with the `my_global.h` file caused compilation of MySQL to fail on single-processor Linux systems running 2.6 kernels. (Bug #10364)
- Temporary tables were created in the data directory instead of `tmpdir` [432]. (Bug #11440)
- A Boolean full-text search where a query contained more query terms than one-third of the query length caused the server to hang or crash. (Bug #7858)
- The `mysqlhotcopy` script was not parsing the output of `SHOW SLAVE STATUS` correctly when called with the `--record_log_pos` option. (Bug #7967)
- Prepared statement with subqueries returned corrupt data. (Bug #11458)
- On Windows, with `lower_case_table_names` [418] set to 2, using `ALTER TABLE` to alter a `MEMORY` or `InnoDB` table that had a mixed-case name also improperly changed the name to lowercase. (Bug #9660)
- `InnoDB` wrongly reported in the `.err` log that MySQL wass trying to drop a nonexistent table, if no more room remained in the tablespace. (Bug #10607)
- `SHOW WARNINGS` did not properly display warnings generated by executing a cached query. (Bug #9414)
- The server could crash due to an attempt to allocate too much memory when `GROUP BY` *`blob_col`* and `COUNT(DISTINCT)` [882] were used. (Bug #11088)
- When applying the `group_concat_max_len` [412] limit, `GROUP_CONCAT()` [883] could truncate multi-byte characters in the middle. (Bug #23451)
- Under certain rare circumstances, inserting into the `mysql.host` table could cause the server to crash. (Bug #10181)
- For `MEMORY` tables, it was possible for updates to be performed using outdated key statistics when the updates involved only very small changes in a very few rows. This resulted in the random failures of queries such as `UPDATE t SET col = col + 1 WHERE col_key = 2;` where the same query with no `WHERE` clause would succeed. (Bug #10178)
- The `--master-data` [296] option for `mysqldump` resulted in no error if the binary log was not enabled. Now an error occurs unless the `--force` option is given. (Bug #11678)
- A `ROLLUP` query could return a wrong result set when its `GROUP BY` clause contained references to the same column. (Bug #11543)
- Testing for `crypt()` support caused compilation problems when using OpenSSL/yaSSL on HP-UX and Mac OS X. (Bug #11150, Bug #10675)
- Manually inserting a row with `host=''` into `mysql.tables_priv` and performing a `FLUSH PRIVILEGES` would cause the server to crash. (Bug #11330)
- Added a missing mutex when rotating the relay logs. Also, the server now logs an error message if the size of a relay log cannot be read. (Bug #6987)

- MySQL would not compile correctly on QNX due to missing `rint()` function. (Bug #11544)
- An incorrect result was obtained for columns that included an aggregate function as part of an expression, and when `WITH ROLLUP` was used with `GROUP BY`. (Bug #7914)
- `INSERT ... SELECT ... ON DUPLICATE KEY UPDATE` produced inaccurate results. (Bug #10886)
- The handling by the `HEX()` [796] function of numbers larger than 2^{64} was improved. (Bug #9854)
- A problem with the `cp1250_czech_cs` collation caused some `LIKE` comparisons to fail. (Bug #9759)
- The value returned by the `FIELD()` [796] function was incorrect when its parameter list contained one or more instances of `NULL`. (Bug #10944)
- The `NULLIF()` [791] function could produce incorrect results if the first argument was `NULL`. (Bug #11142)
- `OPTIMIZE` run on an `InnoDB` table did not return a `Table is full` error if there was insufficient room in the tablespace. (Bug #8135)
- The `mysql` client would output a prompt twice following input of very long strings, because it incorrectly assumed that a call to the `_cgets()` function would clear the input buffer. (Bug #10840)
- Executing `LOAD INDEX INTO CACHE` for a table while other threads where selecting from the table caused a deadlock. (Bug #10602)
- Errors could occur when performing `GROUP BY` on calculated values of a single row table. These could sometimes cause the server to crash on Windows. (Bug #11414)
- Server crashed when using `GROUP BY` on the result of a `DIV` operation on a `DATETIME` value. (Bug #11385)
- Queries against a table using a compound index based on the length of a UTF-8 text column produced incorrect results. For example, given a table with an index defined as shown:

  ```
  CREATE TABLE t (
    id INT NOT NULL,
    city VARCHAR(20) NOT NULL,
    KEY (city(7),id)
  ) TYPE=MYISAM CHARACTER SET=utf8;
  ```

  Assuming that suitable data has been inserted into the table, then a query such as `SELECT * FROM t WHERE city = 'Durban';` would fail. (Bug #10253)
- `GROUP_CONCAT()` [883] with `DISTINCT` and `WITH ROLLUP` ignored `DISTINCT` for some rows. (Bug #7405)
- The `--no-data` [297] option for `mysqldump` was being ignored if table names were given after the database name. (Bug #9558)
- Locking for `CREATE TABLE ... SELECT` for `InnoDB` tables was too weak. It permitted `INSERT` statements issued for the created table while the `CREATE TABLE` statement was still running to appear in the binary log before the `CREATE TABLE` statement. (Bug #6678)
- `SELECT DISTINCT ... GROUP BY` *`constant`* returned multiple rows (it should return a single row). (Bug #8614)
- An overly strict debugging assertion caused debug server builds to fail for some *`col_name = const_expr`*, where *`const_expr`* was a constant expression such as a subquery. (Bug #10020)

- `DROP DATABASE` failed to check for all referencing `InnoDB` tables from other databases before dropping any tables. (Bug #10335)
- `mysqldump` now exports `HASH` index definitions using `USING` rather than `TYPE` when the index name is optional. This corrects a problem when reloading the output for `PRIMARY KEY` definition, because `TYPE` must be preceded an index name, which is not given for a `PRIMARY KEY`. (Bug #11635)
- Using `CONCAT_WS()` [795] on a column set `NOT NULL` caused incorrect results when used in a `LEFT JOIN`. (Bug #11469)
- `SUBSTRING()` [802] did not work properly for input in the `ucs2` character set. (Bug #10344)

C.1.14 Changes in MySQL 4.1.12 (2005-05-13)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

The fix for interpretation of `MERGE` table `.MRG` files (Bug #10687) was made for Windows builds after MySQL 4.1.12 was released and is present in MySQL 4.1.12a.

Functionality Added or Changed

- **Incompatible Change:** The behavior of `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` has changed when the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values both are empty. Formerly, a column was read or written the display width of the column. For example, `INT(4)` was read or written using a field with a width of 4. Now columns are read and written using a field width wide enough to hold all values in the field. However, data files written before this change was made might not be reloaded correctly with `LOAD DATA INFILE` for MySQL 4.1.12 and up. This change also affects data files read by `mysqlimport` and written by `mysqldump --tab`, which use `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`. For more information, see Section 12.2.5, “`LOAD DATA INFILE` Syntax”. (Bug #12564)
- New `/*>` prompt for `mysql`. This prompt indicates that a `/* ... */` comment was begun on an earlier line and the closing `*/` sequence has not yet been seen. (Bug #9186)
- Added a `--debug` [358] option to `my_print_defaults`.
- Updated version of `libedit` to 2.9. (Bug #2596)
- `InnoDB`: When `foreign_key_checks = 0` [411], `ALTER TABLE` and `RENAME TABLE` will ignore any type incompatibilities between referencing and referenced columns. Thus, it will be possible to convert the character sets of columns that participate in a foreign key. Be sure to convert all tables before modifying any data! (Bug #9802)
- `InnoDB`: When the maximum length of `SHOW INNODB STATUS` output would be exceeded, truncate the beginning of the list of active transactions, instead of truncating the end of the output. (Bug #5436)
- When the server cannot read a table because it cannot read the `.frm` file, print a message that the table was created with a different version of MySQL. (This can happen if you create tables that use new features and then downgrade to an older version of MySQL.) (Bug #10435)
- Added the `cp932` Japanese character set.

- Previously in MySQL 4.1, an `Illegal mix of collations` error occurred when mixing strings from same character set when one had a nonbinary collation and the other a binary collation. Now the binary collation takes precedence, so that both strings are treated as having the binary collation. This restores compatibility with MySQL 4.0 behavior.

- `InnoDB`: If `innodb_locks_unsafe_for_binlog` [1070] is enabled and the isolation level of the transaction is not set to `SERIALIZABLE` [976], `InnoDB` uses a consistent read for select in clauses such as `INSERT INTO ... SELECT` and `UPDATE ... (SELECT)` that do not specify `FOR UPDATE` or `LOCK IN SHARE MODE`. Thus, no locks are set to rows read from selected table.

Bugs Fixed

- **Security Fix:** `mysql_install_db` created the `mysql_install_db.X` file with a predictable file name and insecure permissions, which permitted local users to execute arbitrary SQL statements by modifying the file's contents. (CVE-2005-1636)

- **Security Fix:** Starting `mysqld` with `--user=`*`non_existent_user`* [395] caused it to run using the privileges of the account from which it was invoked, including the `root` account. (Bug #9833)

- **Performance:** `InnoDB`: At shutdown, the latest lsn is now written only to the first pages of the `ibdata` files of the system tablespace, and not to the `.ibd` files, saving up to several minutes in some cases.

- **MySQL Cluster:** `AUTO_INCREMENT` did not work with `INSERT..SELECT` on `NDB` tables. (Bug #9675)

- Queries containing `CURRENT_USER()` [872] incorrectly were registered in the query cache. (Bug #9796)

- Concurrent inserts were permitted into the tables in the `SELECT` part of `INSERT ... SELECT ... UNION ...`. This could result in the incorrect order of queries in the binary log. (Bug #9922)

- `myisampack` run on 64-bit systems resulted in segmentation violations. (Bug #9487)

- `InnoDB`: Assertion failures of types `ut_a(cursor->old_stored == BTR_PCUR_OLD_STORED)` and `prebuilt->template_type == 0` could occur when performing multi-table updates. This bug was introduced in 4.1.10 and 4.0.24. (Bug #9670)

- `mysqld` was not checking whether the PID file was successfully created. (Bug #5843)

- `awk` script portability problems were found in `cmd-line-utils/libedit/makelist.sh` . (Bug #9954)

- `SELECT ROUND(`*`expr`*`)` produced a different result from `CREATE TABLE ... SELECT ROUND(`*`expr`*`)`. (Bug #9837)

- `INSERT ... ON DUPLICATE KEY UPDATE` incorrectly updated a `TIMESTAMP` column to the current timestamp, even if the update list included *`col_name = col_name`* for that column to prevent the update. (Bug #7806)

- The `--delimiter` option for the `nds_select` program was nonfunctional. (Bug #10287)

- An error in the implementation of the `MyISAM` compression algorithm caused `myisampack` to fail with very large sets of data (total size of all the records in a single column needed to be at least 3 GB to trigger this issue). (Bug #8321)

- The error message for exceeding `MAX_CONNECTIONS_PER_HOUR` mistakenly referred to `max_connections`. (Bug #9947)

- A problem with `readline`caused the `mysql` client to crash when the user pressed **Control+R**.. (Bug #9568)

- The warning message from `GROUP_CONCAT()` [883] did not always indicate the correct number of lines. (Bug #8681)
- Additional fix for `mysql_server_init()` and `mysql_server_end()` C API functions so that stopping and restarting the embedded server would not cause a crash. (Bug #7344)
- The `latin2_croatian_ci` collation was not sorted correctly. After upgrading to MySQL 4.1.12, all tables that have indexes using this collation are treated as crashed; for each such table, you must use `CHECK TABLE` and possibly repair the table.

  Support for the `cp1250_croatian_ci` collation was also added as part of the fix for this bug. (Bug #6505)
- A deadlock resulted from using `FLUSH TABLES WITH READ LOCK` while an `INSERT DELAYED` statement was in progress. (Bug #7823)
- `InnoDB`: Prevent `ALTER TABLE` from changing the storage engine if there are foreign key constraints on the table. (Bug #5574, Bug #5670)
- The optimizer did not compute the union of two ranges for the `OR` operator correctly. (Bug #9348)
- `ENUM` and `SET` columns in `InnoDB` tables were treated incorrectly as character strings. This bug did not manifest itself with `latin1` collations, but it caused malfunction with `utf8`. Old tables will continue to work. In new tables, `ENUM` and `SET` will be stored internally as unsigned integers. (Bug #9526)
- MySQL no longer automatically blocks IP addresses for which `gethostbyname_r()` fails when the reason is that the DNS server is down. Thanks to Jeremy Cole for patch. (Bug #8467)
- Setting the initial `AUTO_INCREMENT` value for an `InnoDB` table using `CREATE TABLE ... AUTO_INCREMENT = n` did not work, and `ALTER TABLE ... AUTO_INCREMENT = n` did not reset the current value. (Bug #7061)
- `MAX()` [884] for an `INT UNSIGNED` (unsigned 4-byte integer) column could return negative values if the column contained values larger than 2^{31}. (Bug #9298)
- Floats and doubles were not handled correctly when using the prepared statement API in the embedded server. (Bug #10443)
- For a user-defined function invoked from within a prepared statement, the UDF's initialization routine was invoked for each execution of the statement, but the deinitialization routine was not. (It was invoked only when the statement was closed.) For UDFs that have an expensive deinit function (such as `myperl`), this fix has negative performance consequences. (Bug #9913)
- Use of a subquery that used `WITH ROLLUP` in the `FROM` clause of the main query sometimes resulted in a `Column cannot be null` error. (Bug #9681)
- `CAST(string_argument AS UNSIGNED)` [859] didn't work for big integers above the signed range. Now this function and `CAST(string_argument AS SIGNED)` [859] also produces warnings for wrong string arguments. (Bug #7036)
- Memory block allocation did not function correctly for the query cache in the embedded server. (Bug #9549)
- Multiple-table updates could produce spurious data-truncation warnings if they used a join across columns that are indexed using a column prefix. (Bug #9103)
- A deadlock could occur on an update followed by a `SELECT` on an `InnoDB` table without any explicit locks being taken. `InnoDB` now takes an exclusive lock when `INSERT ON DUPLICATE KEY UPDATE` is checking duplicate keys. (Bug #7975)

- `mysql.cc` did not compile correctly using `VC++` on Windows. (Bug #10245)
- `CREATE TABLE ... LIKE` did not work correctly when `lower_case_table_names` [418] was set on a case-sensitive file system and the source table name was not given in lowercase. (Bug #9761)
- Changed metadata for result of `SHOW KEYS`: Data type for `Sub_part` column now is `SMALLINT` rather than `TINYINT` because key part length can be longer than 255. (Bug #9439)
- In the client/server protocol for prepared statements, reconnection failed when the connection was killed with reconnection enabled. (Bug #8866)
- `my_print_defaults` was ignoring the `--defaults-extra-file` [235] option or crashing when the option was given. (Bug #9851, Bug #9136)
- `mysql.server` no longer uses nonportable `alias` command or LSB functions. (Bug #9852)
- `InnoDB`: Crash recovery of `.ibd` files on Windows did not work correctly if `lower_case_table_names = 0`or `lower_case_table_names = 2` had been used; the directory scan used in crash recovery failed to force all paths to lowercase, so that the tablespace name was consistent with the `InnoDB` internal data dictionary.
- `mysqldump` dumped core when invoked with `--tmp` and `--single-transaction` [298] options and a nonexistent table name. (Bug #9175)
- `InnoDB`: Add fault tolerance in the scan of `.ibd` files at a crash recovery; formerly a single failure of `readdir_get_next` caused the rest of the directory to be skipped.
- An `InnoDB` test suite failure was caused by a locking conflict between two server instances at server shutdown or startup. This conflict on advisory locks appears to be the result of a bug in the operating system; these locks should be released when the files are closed, but somehow that does not always happen immediately in Linux. (Bug #9381)
- When `SELECT constant` was the final `SELECT` in a `UNION`, a trailing `LIMIT ...` worked, but a trailing `ORDER BY ...` or `ORDER BY ... LIMIT ...` did not. (Bug #10032)
- `CHAR` and `VARCHAR` columns that used the `sjis` character set were not being saved correctly, causing the following columns to be corrupted. (Bug #10493)
- A server installed as a Windows service and started with `--shared-memory` [392] could not be stopped. (Bug #9665)
- Extraneous comparisons between `NULL` values in indexed columns were performed by the optimzer for operators such as `=` that are never true for `NULL`. (Bug #8877)
- In some cases, concurrent `DELETE` and `INSERT...SELECT` queries could crash the MySQL server. The issue was a problem in the key cache. (Bug #10167)
- For `MERGE` tables, avoid writing absolute path names in the `.MRG` file for the names of the constituent `MyISAM` tables so that if the data directory is moved, `MERGE` tables will not break. For `mysqld`, write just the `MyISAM` table name if it is in the same database as the `MERGE` table, and a path relative to the data directory otherwise. For the embedded servers, absolute path names may still be used. (Bug #5964)
- Indexes on `MyISAM` tables could sometimes be corrupted; this was the result of padding values with spaces for comparison: Dumping a table with `mysqldump`, reloading it, and then re-running the binary log against it crashed the index and required a repair. (Bug #9188)
- With `DISTINCT`, `CONCAT(col_name,...)` [795] returned incorrect results when the arguments to `CONCAT()` [795] were columns with an integer data type declared with a display width narrower than the values in the column. (For example, if an `INT(1)` column contained `1111`.) (Bug #4082)

- `RENAME TABLE` for an `ARCHIVE` table failed if the `.arn` file was not present. (Bug #9911)
- An error occurred if you specified a default value of `TRUE` or `FALSE` for a `BOOL` column. (Bug #9666)
- Starting `mysqld` with the `--skip-innodb` [1066] and `--default-storage-engine=innodb` [386] (or `--default-table-type=innodb` [386] caused a server crash. (Bug #9815)
- `MERGE` tables could fail on Windows due to incorrect interpretation of path name separator characters for file names in the `.MRG` file. (Bug #10687)
- `net_read_timeout` [422] and `net_write_timeout` [422] were not being respected on Windows. (Bug #9721)
- `TIMEDIFF()` [840] with a negative time first argument and positive time second argument produced incorrect results. (Bug #8068)
- A segmentation fault in `mysqlcheck` occurred when the last table checked in `--auto-repair` [285] mode returned an error (such as the table being a `MERGE` table). (Bug #9492)
- Remove extra slashes in `--tmpdir` value (for example, convert `/var//tmp` to `/var/tmp`, because they caused various errors. (Bug #8497)
- Corrected some failures of prepared statements for SQL (`PREPARE` plus `EXECUTE`) to return all rows for some `SELECT` statements. (Bug #9777, Bug #9096)
- The server did not compile correctly with MinGW. Our thanks to Nils Durner for the patch. (Bug #8872)
- `configure` did not properly recognize whether NPTL was available on Linux. (Bug #2173)
- `configure` did not check the system for atomic operations capabilities. (Bug #7970)

C.1.15 Changes in MySQL 4.1.11 (2005-04-01)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- **MySQL Cluster; Replication:** Added a new global system variable `slave_transaction_retries` [1181]: If the replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or exceeded `InnoDB`'s `innodb_lock_wait_timeout` [1069] or `NDBCLUSTER`'s `TransactionDeadlockDetectionTimeout` [1269] or `TransactionInactiveTimeout` [1269], it automatically retries `slave_transaction_retries` [1181] times before stopping with an error. The default is 0, and you must explicitly set the value greater than 0 to enable the “retry” behavior. (Bug #8325)
- **MySQL Cluster:** More informative error messages are provided when a query is issued against an `NDB` table that has been modified by another `mysqld` server. (Bug #6762)
- **Replication:** For slave replication servers started with `--replicate-*` options, statements that should not be replicated according those options no longer are written to the slave's general query log. (Bug #8297)
- `NULL` now is considered more coercible than string constants. This resolves some `Illegal mix of collations` conflicts.

- Added configuration directives `!include` and `!includedir` for including option files and searching directories for option files. See Section 4.2.3.3, “Using Option Files”, for usage.
- Modified the parser to permit `SELECT` statements following the `UNION` keyword to be subqueries in parentheses. (Bug #2435)
- Added `sql_notes` [429] session variable to cause `Note`-level warnings not to be recorded. (Bug #6662)
- The coercibility for the return value of functions such as `USER()` [876] or `VERSION()` [876] now is “system constant” rather than “implicit.” This makes these functions more coercible than column values so that comparisons of the two do not result in `Illegal mix of collations` errors. `COERCIBILITY()` [871] was modified to accommodate this new coercibility value. See Section 11.13, “Information Functions”.
- Added `--with-big-tables` [98] compilation option to `configure`. (Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually to enable large table support.) See Section 2.9.3, “MySQL Source-Configuration Options”, for details.
- User variable coercibility has been changed from “coercible” to “implicit.” That is, user variables have the same coercibility as column values.
- The use of `SESSION` or `GLOBAL` is no longer permitted for user variables. (Bug #9286)
- `InnoDB`: Commit after every 10,000 copied rows when executing `CREATE INDEX`, `DROP INDEX` or `OPTIMIZE TABLE`, which are internally implemented as `ALTER TABLE`. This makes it much faster to recover from an aborted operation.
- `mysqld_safe` will create the directory where the UNIX socket file is to be located if the directory does not exist. This applies only to the last component of the directory path name. (Bug #8513)
- `ONLY_FULL_GROUP_BY` [460] no longer is included in the `ANSI` [460] composite SQL mode. (Bug #8510)

Bugs Fixed

- **Replication:** If multiple semicolon-separated statements were received in a single packet, they were written to the binary log as a single event rather than as separate per-statement events. For a server serving as a replication master, this caused replication to fail when the event was sent to slave servers. (Bug #8436)
- **Replication:** A replication master stamped a generated statement (such as a `SET` statement) with an error code intended only for another statement. This could happen, for example, when a statement generated a duplicate key error on the master but still had be to replicated to the slave. (Bug #8412)
- **Replication:** Treat user variables as having `IMPLICIT` derivation (coercibility) to avoid “Illegal mix of collations” errors when replicating user variables. (Bug #6676)
- **Replication:** If the slave was running with `--replicate-*-table` options which excluded one temporary table and included another, and the two tables were used in a single `DROP TEMPORARY TABLE IF EXISTS` statement, as the ones the master automatically writes to its binary log upon client's disconnection when client has not explicitly dropped these, the slave could forget to delete the included replicated temporary table. Only the slave needs to be upgraded. (Bug #8055)
- The `tee` command could sometimes cause the `mysql` client to crash. (Bug #8499)
- `MATCH ... AGAINST` in natural language mode could cause a server crash if the `FULLTEXT` index was not used in a join (that is, `EXPLAIN` did not show `fulltext` [567] join mode) and the search query matched no rows in the table. (Bug #8522)

- Unions between binary and nonbinary columns failed due to a collation coercibility problem. (Bug #6519)
- Conversion of strings to doubles is now more accurate for floating point values that can be represented by integers, such as `123.45E+02`. (Bug #7840)
- `REPAIR TABLE` did not invalidate query results in the query cache that were generated from the table. (Bug #8480)
- Using `NOW()` [837] in a subquery caused the server to crash. (Bug #8824)
- `InnoDB`: Work around a problem in AIX 5.1 patched with ML7 security patch: `InnoDB` would refuse to open its `ibdata` files, complaining about an operating system error 0.
- If the `mysql` prompt was configured to display the default database name, and that database was dropped, `mysql` did not update the prompt. (Bug #4802)
- Use of `GROUP_CONCAT(x)` [883] in a subquery, where *x* was an alias to a column in the outer query, resulted in a server crash. (Bug #8656)
- For a query with both `GROUP BY` and `COUNT(DISTINCT)` [882] clauses and a `FROM` clause with a subquery, `NULL` was returned for any `VARCHAR` column selected by the subquery. (Bug #8218)
- Changed `mysql_server_end()` C API function to restore more variables to their initial state so that a subsequent call to `mysql_server_init()` would not cause a client program crash. (Bug #7344)
- Setting the `max_error_count` [419] system variable to 0 resulted in a setting of 1. (Bug #9072)
- In prepared statements, subqueries containing parameters were erroneously treated as `const` [567] tables during preparation, resulting in a server crash. (Bug #8807)
- `InnoDB`: Honor the `--tmpdir` [394] startup option when creating temporary files. Previously, `InnoDB` temporary files were always created in the temporary directory of the operating system. On Netware, `InnoDB` will continue to ignore `--tmpdir` [394]. (Bug #5822)
- The `--set-character-set` option for `myisamchk` was changed to `--set-collation` [318]. The value needed for specifying how to sort indexes is a collation name, not a character set name. (Bug #8349)
- With `lower_case_table_names` [418] set to 1, `mysqldump` on Windows could write the same table name in different lettercase for different SQL statements. (Bug #8216)
- A rare race condition could cause `FLUSH TABLES WITH READ LOCK` to hang. (Bug #8682)
- Using a compariosn where the left expression of `IN`, `ALL`, or `ANY` was a subquery caused the server to crash (Bug #8888)
- Neither `SHOW ERRORS` nor `SHOW WARNINGS` were displaying Error-level messages. (Bug #6572)
- When the server was started with `--skip-name-resolve` [393], specifying host name values that included netmasks in `GRANT` statements did not work. (Bug #8471)
- Table creation for a `MyISAM` table failed if `DATA DIRECTORY` or `INDEX DIRECTORY` options were given that specified the path name to the database directory where the table files would be created by default. (Bug #8707)
- Matching of table names by `mysqlhotcopy` now accommodates `DBD::mysql` versions 2.9003 and up, which implement identifier quoting. (Bug #8136)
- `mysqldump` misinterpreted "_" and "%" characters in the names of tables to be dumped as wildcard characters. (Bug #9123)

- On Windows, create shared memory objects with the proper access rights to make them usable when the client and server are running under different accounts. (Bug #8226)
- InnoDB: If one used LOCK TABLES, created an InnoDB temp table, and did a multiple-table update where a MyISAM table was the update table and the temp table was a read table, then InnoDB asserted in row0sel.c because n_mysql_tables_in_use was 0. Also, we remove the assertion altogether and just print an error to the .err log if this important consistency check fails. (Bug #8677)
- Accented letters were improperly treated as distinct by the utf_general_ci collation. (Bug #7878)
- Using TIMESTAMP columns with no minute or second parts in GROUP BY clauses with the new [390] system variable set to 1 caused the server to crash. (Bug #9401)
- The utf8_spanish2_ci and ucs2_spanish2_ci collations no longer consider r equal to rr . If you upgrade to this version from an earlier version, you should rebuild the indexes of any affected tables. (Bug #9269)
- Privileges could be escalated using database wildcards in GRANT statements. (CVE-2004-0957)
- Made the relay_log_space_limit [426] system variable show up in the output of SHOW VARIABLES. (Bug #7100)
- my_print_defaults ignored the --defaults-extra-file [235] and --defaults-file [235] options.
- Some user variables were not being handled with “implicit” coercibility. (Bug #9425)
- mysqldump now avoids writing SET NAMES to the dump output if the server is older than version 4.1 and would not understand that statement. (Bug #7997)
- Worked around a bug in support for NSS support in glibc when static linking is used and LDAP is one of the NSS sources. The workaround is to detect when the bug causes a segmentation fault and issue a diagnostic message with information about the problem. (Bug #4872, Bug #3037)
- An expression that tested a case-insensitive character column against string constants that differed in lettercase could fail because the constants were treated as having a binary collation. (For example, WHERE city='London' AND city='london' could fail.) (Bug #7098, Bug #8690)
- InnoDB: If InnoDB cannot allocate memory, keep retrying for 60 seconds before we intentionally crash mysqld; maybe the memory shortage is just temporary.
- When using the cp1250_czech_cs collation, empty literal strings were not regarded as equal to empty character columns. (Bug #8840)
- InnoDB: If MySQL wrote to its binlog, but for some reason trx->update_undo and trx->insert_undo were NULL in InnoDB, then trx->commit_lsn was garbage, and InnoDB could assert in the log flush of trx_commit_complete_for_mysql(). (Bug #9277)
- When setting integer system variables to a negative value with SET VARIABLES, the value was treated as a positive value modulo 2^{32}. (Bug #6958)
- Too many rows were returned from queries that combined ROLLUP and LIMIT if SQL_CALC_FOUND_ROWS was given. (Bug #8617)
- A problem with static variables did not permit building the server on Fedora Core 3. (Bug #6554)
- Platform and architecture information in version information produced for --version option on Windows was always Win95/Win98 (i32). More accurately determine platform as Win32 or Win64

for 32-bit or 64-bit Windows, and architecture as `ia32` for x86, `ia64` for Itanium, and `axp` for Alpha. (Bug #4445)

- The use of `XOR` together with `NOT ISNULL()` erroneously resulted in some outer joins being converted to inner joins by the optimizer. (Bug #9017)
- Subqueries using `ALL`or `ANY` that contained a `HAVING` clause did not work correctly. (Bug #9350)
- The `MAX_CONNECTIONS_PER_HOUR` resource limit was not being reset hourly and thus imposed an absolute limit on number of connections per account until the server is restarted or the limits flushed. (Bug #8350)
- `LIKE` pattern-matching for strings did not work correctly with the `cp1251_bin` collation. (Bug #8560)
- Expressions involving nested `CONCAT()` [795] calls and character set conversion of string constants could return an incorrect result. (Bug #8785)
- Creating a table using a name containing a character that is illegal in `character_set_client` [408] resulted in the character being stripped from the name and no error. The character now is considered an error. (Bug #8041)
- Host name matching didn't work if a netmask was specified for table-specific privileges. (Bug #3309)
- Binary data stored in `BLOB` or `BINARY` columns would be erroneously dumped if `mysqldump` was invoked with `--hex-blob` [295] and `--skip-extended-insert` arguments. This happened if data contained characters larger then 0x7F . (Bug #8830)
- Mixed-case database and table names in the grant tables were ignored for authentication if the `lower_case_table_names` [418] system variable was set. `GRANT` will not create such privileges when `lower_case_table_names` [418] is set, but it is possible to create them by direct manipulation of the grant tables, or that old grant records were present before setting the variable. (Bug #7989)
- The bundled `readline` library caused a segmentation fault in `mysql` when the user entered **Shift**+**Enter**. (Bug #5672)
- Killing a filesort could cause an assertion failure. (Bug #8799)
- Permit extra HKSCS and cp950 characters (`big5` extension characters) to be accepted in `big5` columns. (Bug #9357)
- Do not try to space-pad `BLOB` columns containing `ucs2` characters. (Bug #8771)

  References: This bug was introduced by Bug #7350.
- Incorrectly ordered results were returned from a query using a `FULLTEXT` index to retrieve rows and there was another index that was usable for `ORDER BY`. For such a query, `EXPLAIN` showed the `fulltext` [567] join type, but showed the other (not `FULLTEXT`) index in the `Key` column. (Bug #6635)
- The `CHARSET()` [870], `COLLATION()` [871], and `COERCIBILITY()` [871] functions sometimes returned `NULL`. `CHARSET()` [870] and `COLLATION()` [871] returned `NULL` when given any of these arguments that evaluated to `NULL`: A system function such as `DATABASE()` [872]; a column value; and a user variable. Now `CHARSET()` [870] and `COLLATION()` [871] return the system character set and collation; the column character set and collation; and `binary`. `COERCIBILITY(NULL)` [871] now returns “ignorable” coercibility rather than `NULL`. (Bug #9129)
- Fixed option-parsing code for the embedded server to understand `K`, `M`, and `G` suffixes for the `net_buffer_length` [422] and `max_allowed_packet` [418] options. (Bug #9472)

- `FOUND_ROWS()` [872] returned an incorrect value for preceding `SELECT` statements that used no table or view. (Bug #6089)
- Incorrect results were returned from queries that combined `SELECT DISTINCT`, `GROUP BY` , and `ROLLUP`. (Bug #8616)
- When performing boolean full-text searches on `utf8` columns, a double-quote character in the search string caused the server to crash. (Bug #8351)
- A problem in index cost calculation caused a `USE INDEX` or `FORCE INDEX` hint not to be used properly for a `LEFT JOIN` across indexed `BLOB` columns. (Bug #7520)
- In string literals with an escape character (“`\`”) followed by a multi-byte character that has a second byte of “`\`”, the literal was not interpreted correctly. The next character now is escaped, not just the next byte. (Bug #8303)
- `perror` was printing a spurious extra line of output ("Error code ###: Unknown error ###" printed directly before the correct line with the error message). (Bug #8517)
- `OPTIMIZE TABLE` was written twice to the binary log when used on `InnoDB` tables. (Bug #9149)
- Ordering by an unsigned expression (more complex than a column reference) was treating the value as signed, producing incorrectly sorted results. `HAVING` was also treating unsigned columns as signed. (Bug #7425)
- The `MEMORY` storage engine did not properly increment an `AUTO_INCREMENT` column if there was a second composite index that included the column. (Bug #8489)
- The output of the `STATUS` (`\s`) command in `mysql` had the values for the server and client character sets reversed. (Bug #7571)
- `ENUM` and `SET` columns in privilege tables incorrectly had a case-sensitive collation, resulting in failure of assignments of values that did not have the same lettercase as given in the column definitions. The collation was changed to be case insensitive. (Bug #7617)
- `InnoDB`: A table with a primary key that contained at least two column prefixes was prone to memory corruption. An example of an affected `CREATE TABLE` statement is shown here:

```
CREATE TABLE t  (
    a CHAR(100),
    b TINYBLOB,
    PRIMARY KEY(a(5), b(10))
);
```

- If a `MyISAM` table on Windows had `INDEX DIRECTORY` or `DATA DIRECTORY` table options, `mysqldump` dumped the directory path names with single-backslash path name separators. This would cause syntax errors when importing the dump file. `mysqldump` now changes “`\`” to “`/`” in the path names on Windows. (Bug #6660)
- For a statement string that contained multiple slow queries, only the last one would be written to the slow query log. (Bug #8475)
- `MIN(col_name)` [884] and `MAX(col_name)` [884] could fail to produce the correct result if *`col_name`* was contained in multiple indexes and the optimizer did not choose the first index that contained the column. (Bug #8893)
- For `MyISAM` tables, `REPAIR TABLE` no longer discard rows that have incorrect checksum. (Bug #9824)

- The data type for `MAX(datetime_col)` [884] was returned as `VARCHAR` rather than `DATETIME` if the query included a `GROUP BY` clause. (Bug #5615)
- Depending on index statistics, `GROUP BY col1, col2, ...` could return incorrect results if the first table processed for a join had several indexes that cover the grouped columns. (Bug #9213)
- A join on two tables failed when each contained a `BIGINT UNSIGNED` column that were compared when their values exceeded $2^{63} - 1$. The match failed and the join returned no rows. (Bug #8562)
- `BLOB(M)` and `TEXT(M)` columns, with `M` less than 256, were being created as `BLOB` and `TEXT` columns rather than `TINYBLOB` or `TINYTEXT` columns. (Bug #9303)
- `AES_DECRYPT(col_name,key)` [865] could fail to return `NULL` for invalid values in `col_name`, if `col_name` was declared as `NOT NULL`. (Bug #8669)
- `InnoDB`: An error in `mysqld` caused `InnoDB` in MySQL 4.1.8 to 4.1.10 `InnoDB` to refuse to use a table created with MySQL 3.23.49 or earlier if it was in the new compact `InnoDB` table format of 5.0.3 or later.

  Workaround. Upgrade to 4.1.11 or newer, or dump the table and re-create it with MySQL 3.23.50 or newer before upgrading.
- If `max_join_size` [419] was set, a query containing a subquery that exceeded the examined-rows limit could hang. (Bug #8726)
- With a database was dropped with `lower_case_table_names = 2` [418], tables in the database also were dropped but not being flushed properly from the table cache. If the database was re-created, the tables also would appear to have been re-created. (Bug #8355)
- Retrieving from a view defined as a `SELECT` that mixed `UNION ALL` and `UNION DISTINCT` resulted in a different result than retrieving from the original `SELECT`. (Bug #6565)
- Corruption of `MyISAM` table indexes could occur with `TRUNCATE TABLE` if the table had already been opened. For example, this was possible if the table had been opened implicitly by selecting from a `MERGE` table that mapped to the `MyISAM` table. The server now issues an error message for `TRUNCATE TABLE` under these conditions. (Bug #8306)
- The Cyrillic letters `I` (И) and `SHORT I` (Й) were treated as being the same character by the `utf8_general_ci` collation. (Bug #8385)
- Queries that combined `SELECT DISTINCT`, `SUM()` [884], and `ROLLUP` could cause the MySQL server to crash. (Bug #8615)
- The `CHAR()` [794] function was not ignoring `NULL` arguments, contrary to the documentation. (Bug #6317)
- Bundled `zlib` in the source distribution was upgraded to 1.2.2. (Bug #9118)

C.1.16 Changes in MySQL 4.1.10 (2005-02-12)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

The security improvements related to creation of table files and to user-defined functions were made after MySQL 4.1.10 was released and are

present in MySQL 4.1.10a. We would like to thank Stefano Di Paola <stefano.dipaola@wisec.it> for making us aware of these.

Functionality Added or Changed

- Setting the connection collation to a value different from the server collation followed by a `CREATE TABLE` statement that included a quoted default value resulted in a server crash. (Bug #8235)
- Added `mysql_library_init()` and `mysql_library_end()` as synonyms for the `mysql_server_init()` and `mysql_server_end()` C API functions. `mysql_library_init()` and `mysql_library_end()` are `#define` symbols, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`. (Bug #6149)
- `InnoDB`: A shared record lock (`LOCK_REC_NOT_GAP`) is now taken for a matching record in the foreign key check because inserts can be permitted into gaps.
- Thread stack size was increased from 192KB to 256KB on Linux/IA-64 (too small stack size was causing server crashes on some queries). (Bug #8391)
- The server now issues a warning when `lower_case_table_names = 2` [418] and the data directory is on a case-sensitive file system, just as when `lower_case_table_names = 0` [418] on a case-insensitive file system. (Bug #7887)
- Security improvement: The server creates `.frm`, `.MYD`, `.MYI`, `.MRG`, `.ISD`, and `.ISM` table files only if a file with the same name does not already exist. Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and informing us about this issue. (CVE-2005-0711)
- Added back faster subquery execution from 4.1.8. This adds also back a bug from 4.1.8 in comparing `NULL` to the value of a subquery.
- Security improvement: User-defined functions should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. `mysqld` by default no longer loads UDFs unless they have at least one auxiliary symbol defined in addition to the main symbol. The `--allow-suspicious-udfs` [384] option controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off. `mysqld` also checks UDF file names when it reads them from the `mysql.func` table and rejects those that contain directory path name separator characters. (It already checked names as given in `CREATE FUNCTION` statements.) See Section 18.2.2.1, “UDF Calling Sequences for Simple Functions”, Section 18.2.2.2, “UDF Calling Sequences for Aggregate Functions”, and Section 18.2.2.6, “User-Defined Function Security Precautions”. Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and informing us about this issue. (CVE-2005-0709, CVE-2005-0710)
- `InnoDB`: When MySQL/InnoDB is compiled on Mac OS X 10.2 or earlier, detect the operating system version at run time and use the `fcntl()` file flush method on Mac OS X versions 10.3 and later. In Mac OS X, `fsync()` does not flush the write cache in the disk drive, but the special `fcntl()` does; however, the flush request is ignored by some external devices. Failure to flush the buffers may cause severe database corruption at power outages.
- From the Windows distribution, predefined accounts without passwords for remote users (`'root'@'%'`, `''@'%'`) were removed (other distributions never had them).
- `InnoDB`: Relaxed locking in `INSERT ... SELECT`, single table `UPDATE ... (SELECT)` and single table `DELETE ... (SELECT)` clauses when `innodb_locks_unsafe_for_binlog` [1070] is used and isolation level of the transaction is not `SERIALIZABLE` [976]. `InnoDB` uses consistent read in these cases for a selected table.

- The server now issues a warning to the error log when it encounters older tables that contain character columns that might be interpreted by newer servers to have a different column length. See Section 2.11.1.1, “Upgrading from MySQL 4.0 to 4.1”, for a discussion of this problem and what to do about it. (Bug #6913)

Bugs Fixed

- **Replication:** Multiple-table updates did not replicate properly to slave servers where `--replicate-*-table` options had been specified. (Bug #7011)
- The `CONVERT_TZ()` [828] function, when its second or third argument was from a `const` [567] table, caused the server to crash. (See Section 12.7.2, “`EXPLAIN` Syntax”.) (Bug #7705)
- `FOUND_ROWS()` [872] returned an incorrect value after a `SELECT SQL_CALC_FOUND_ROWS DISTINCT` statement that selected constants and included `GROUP BY` and `LIMIT` clauses. (Bug #7945)
- The `CONV()` [818] function returned an unsigned `BIGINT` number, which does not fit in 32 bits. (Bug #7751)
- `TIMESTAMP` columns with their display width so specified were not treated as identical to `DATETIME` columns when the server was run in `MAXDB` [460] mode. (Bug #7418)
- MySQL permitted concurrent updates (including inserts and deletes) to a table if binary logging was enabled. Now, all updates are executed in a serialized fashion, because they are executed serialized when the binary log is replayed. (Bug #7879)
- The `TIMEDIFF()` [840] function returned incorrect results if one of its arguments had a nonzero microsecond part. (Bug #7586)
- `InnoDB`: `ALTER TABLE ... ADD CONSTRAINT PRIMARY KEY ...` complained about bad foreign key definition. (Bug #7831)
- Updates were being written to the binary log when there were `binlog-do-db` or `binlog-ignore-db` options even when there was no current database, contrary to Section 14.9.1, “Evaluation of Database-Level Replication and Binary Logging Options”. (Bug #6749)
- `SHOW INDEX` on a `MERGE` table could cause debug versions of the server to crash. (Bug #7377)
- The number of columns in a row comparison against a subquery was calculated incorrectly. (Bug #8020)
- Conversion of floating-point values to character values was not performed correctly when the absolute value of the float was less than 1 (including negative values). (Bug #7774)
- For indexes, `SHOW CREATE TABLE` now displays the index type even if it is the default, for storage engines that support multiple index types. (Bug #7235)
- A slave running MySQL 3.23.51 or newer hung while trying to connect to a master running MySQL 3.23.50 or older. (This occurred due to a bug in the old masters—`SELECT @@unknown_var` caused the server to hang—which was fixed in MySQL 3.23.50.) (Bug #7965)
- `mysqld` had problems finding its language files if the `--basedir` [384] value was specified as a very long path name. (Bug #8015)
- `InnoDB`: A rare race condition could cause an assertion in `DROP TABLE` or in `ALTER TABLE`.
- `ALTER TABLE` on a `TEMPORARY` table with a mixed-lettercase name could cause the table to disappear when `lower_case_table_names` [418] was set to 2. (Bug #7261)

- Multiple-table `UPDATE` statements could cause spurious `Table '#sql_....' is full` errors if the number of rows to update was sufficiently large. (Bug #7788)
- `LOAD INDEX` statement now loads the index into memory. (Bug #8452)
- Corrected a problem with references to `DUAL` where statements such as `SELECT 1 AS a FROM DUAL` would succeed but statements such as `SELECT 1 AS a FROM DUAL LIMIT 1` would fail. (Bug #8023)
- Strings that began with `CHAR(31)` were considered equal to the empty string. (Bug #8134)
- Executing a multi-statement query more than once with the query cache active could yield incorrect result sets. (Bug #7966)
- `InnoDB`: Fixed a bug introduced in 4.1.9, where, if you used `innodb_file_per_table` [1068] with the Windows version of MySQL, `mysqld` stopped with Windows error 87. (See the Bugs database or the MySQL 4.1.9 changelog for information about a workaround for the issue in 4.1.9). (Bug #8021)
- If one used `CONVERT_TZ()` [828] function in `SELECT`, which in its turn was used in `CREATE TABLE` statements, then system time zone tables were added to list of tables joined in SELECT and thus erroneous result was produced. (Bug #7899)
- If multiple prepared statements were executed without retrieving their results, executing one of them again would cause the client program to crash. (Bug #8330)
- The `IN()` [784] operator did not return correct results if all values in the list were constants and some of them used substring functions such as `LEFT()` [797], `RIGHT()` [800], or `MID()` [799]. (Bug #7716)
- Nonnumeric values inserted into a `YEAR` column were being stored as `2000` rather than as `0000`. (Bug #6067)
- The combination of `-not` and `trunc*` operators in a full-text search did not work correctly. Using more than one truncated negative search term caused the result to be empty.
- `SHOW INDEX` reported `Sub_part` values in bytes rather than characters for columns with a multi-byte character set. (Bug #7943)
- Adding an `ORDER BY` clause for an indexed column caused a `SELECT` to return an empty result. (Bug #7331)
- `InnoDB`: Use native `tmpfile()` function on Netware. All `InnoDB` temporary files are created under `sys:\tmp`. Previously, `InnoDB` temporary files were never deleted on Netware.
- `CREATE TABLE ... LIKE` failed on Windows when the source or destination table was located in a symlinked database directory. (Bug #6607)
- Re-execution of prepared statements containing subqueries caused the server to crash. (Bug #8125)
- `ALTER TABLE` improperly accepted an index on a `TIMESTAMP` column that `CREATE TABLE` would reject. (Bug #7884)
- Handling of trailing spaces was incorrect for the `ucs2` character set. (Bug #7350)
- Certain correlated subqueries with forward references (referring to an alias defined later in the outer query) could crash the server. (Bug #8025)
- Key cache statistics were reported incorrectly by the server after receipt of a `SIGHUP` signal. (Bug #4285)

- Correct a problem with `mysql_config`, which was failing to produce proper `zlib` option for linking under some circumstances. (Bug #6273)
- Comparing a nested row expression (such as `ROW(1,(2,3))`) with a subquery caused the server to crash. (Bug #8022)
- `mysqlbinlog` forgot to add backquotes around the collation of user variables (causing later parsing problems as `BINARY` is a reserved word). (Bug #7793)
- A symlink vulnerability in the `mysqlaccess` script was reported by Javier Fernandez-Sanguino Pena and Debian Security Audit Team. (CVE-2005-0004)
- A `HAVING` clause that referred to `RAND()` [822] or a user-defined function in the `SELECT` part of a query through an alias could cause MySQL to crash or to return an incorrect value. (Bug #5185)
- Erroneous output resulted from `SELECT DISTINCT` combined with a subquery and `GROUP BY`. (Bug #7946)
- Column headers in query results retrieved from the query cache could be corrupted when a non-4.1 client was served a result originally generated for a 4.1 client. The query cache was not keeping track of which client/server protocol was being used. (Bug #6511)
- Modify `SET` statements produced by `mysqldump` to write quoted strings using single quotation marks rather than double quotation marks. This avoids problems if the dump file is reloaded while the `ANSI_QUOTES` [458] SQL mode is in effect. (Bug #8148)
- Changed `mysql` client so that including `\p` as part of a prompt command uses the name of the shared memory connection when the connection is using shared memory. (Bug #7922)
- Cardinality estimates for `HASH` indexes of `TEMPORARY` tables created using `MEMORY` storage engine were inaccurate. As a result, queries that were using this index (as shown by `EXPLAIN`) could returned incorrect results. (Bug #8371)
- Add description of `debug` command to `mysqladmin` help output. (Bug #8207)
- A problem with `UNION` statements resulted in the wrong number of examined rows being reported in the slow query log.
- `DELETE FROM tbl_name ... WHERE ... ORDER BY tbl_name.col_name` when the `ORDER BY` column was qualified with the table name caused the server to crash. (Bug #8392)
- `mysql_stmt_close()` C API function was not clearing an error indicator when a previous prepare call failed, causing subsequent invocations of error-retrieving calls to indicate spurious error values. (Bug #7990)
- `mysql_stmt_prepare()` was very slow when used in client programs on Windows. (Bug #5787)
- A `Table is full` error occurred when the table was still smaller than `max_heap_table_size` [419]. (Bug #7791)
- `perror.exe` was always returning “Unknown error” on Windows. See Section 4.8.1, “`perror` — Explain Error Codes”. (Bug #7390)
- Removed a dependence of boolean full-text search on `--default-character-set` option. (Bug #8159)
- Comparing the result of a subquery to a nonexistent column caused the server to crash. This issue affected MySQL on Windows platforms only. (Bug #7885)

- Use of `GROUP_CONCAT()` [883] with `HAVING` caused the server to crash. (Bug #7769)
- Certain joins used with boolean full-text search could cause the server to crash. (Bug #8234)
- Ensured that `mysqldump --single-transaction` sets its transaction isolation level to `REPEATABLE READ` [976] before proceeding (otherwise if the MySQL server was configured to run with a default isolation level lower than `REPEATABLE READ` [976] it could give an inconsistent dump). (Bug #7850)

C.1.17 Changes in MySQL 4.1.9 (2005-01-11)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- `mysqld_safe` no longer tests for the presence of the data directory when using a relatively located server binary. It just assumes the directory is there, and fails to start up if it is not. This permits the data directory location to be specified on the command line, and avoids running a server binary that was not intended. (Bug #7249)

  References: See also Bug #7518.
- The MySQL server aborts immediately instead of simply issuing a warning if it is started with the `--log-bin` [1181] option but cannot initialize the binary log at startup (that is, an error occurs when writing to the binary log file or binary log index file).
- The platform suffix was changed from `-win` to `-win32`
- `InnoDB`: Print a more descriptive error and refuse to start `InnoDB` if the size of `ibdata` files is smaller than what is stored in the tablespace header; `innodb_force_recovery` [1069] overrides this.
- The `MySQL-shared-compat` Linux RPM now includes the 3.23 as well as the 4.0 `libysqlclient.so` shared libraries. (Bug #6342)
- The product descriptions `-noinstall` and `-essential` have been moved in front of the version number
- The binary log file and binary log index file now behave like `MyISAM` when there is a "disk full" or "quota exceeded" error. See Section B.5.4.3, “How MySQL Handles a Full Disk”.
- `InnoDB`: Do not acquire an internal `InnoDB` table lock in `LOCK TABLES` if `autocommit = 1` [406]. This helps in porting old `MyISAM` applications to `InnoDB`. `InnoDB` table locks in that case caused deadlocks very easily.
- `Seconds_Behind_Master` is `NULL` (which means “unknown”) if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. It is zero if the SQL thread has caught up with the I/O thread. It no longer grows indefinitely if the master is idle.
- The naming scheme of the Windows installation packages has changed slightly:
  - The platform suffix was changed from `-win` to `-win32`
  - The product descriptions `-noinstall` and `-essential` have been moved in front of the version number

  Examples: `mysql-essential-4.1.9-win32.msi`, `mysql-noinstall-4.1.9-win32.zip` See Section 2.3, “Installing MySQL on Microsoft Windows”.

- The Mac OS X 10.3 installation disk images now include a MySQL Preference Pane for the Mac OS X Control Panel that enables the user to start and stop the MySQL server using the GUI and activate and deactivate the automatic MySQL server startup on bootup.

Bugs Fixed

- **Replication:** A replication slave could crash after replicating many `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `REPAIR TABLE` statements from the master. (Bug #6461, Bug #7658)
- `FLOAT` values were not truncated correctly. (Bug #7361)
- When encountering a `disk full` or `quota exceeded` write error, `MyISAM` sometimes failed to sleep and retry the write, resulting in a corrupted table. (Bug #7714)
- Include compression library flags in the output from `mysql_config --lib_r`. (Bug #7021)
- Made the MySQL server accept executing `SHOW CREATE DATABASE` even if the connection has an open transaction or locked tables. Refusing it made `mysqldump --single-transaction` sometimes fail to print a complete `CREATE DATABASE` statement for some dumped databases. (Bug #7358)
- Corrected a problem with `mysql_config` not producing all relevant flags from `CFLAGS`. (Bug #6964)
- `mysqladmin password` now checks whether the server has the `old_passwords` [423] enabled or predates 4.1 and uses the old-format password if so. (Bug #7451)
- Running `mysql_fix_privilege_tables` could result in grant table columns with lengths that were too short if the server character set had been set to a multi-byte character set first. (Bug #7539)
- Incorrect results were obtained for complex datetime expressions containing casts of datetime values to `TIME` or `DATE` values. (Bug #6914)
- `InnoDB`: `InnoDB` failed to drop a table in the background drop queue if the table was referenced by a `FOREIGN KEY` constraint.
- `PROCEDURE ANALYSE()` did not quote some `ENUM` values properly. (Bug #2813)
- Using `INSERT DELAYED` with prepared statements could lead to table corruption.
- `InnoDB`: When `DISCARD TABLESPACE` failed because the table was referenced by a foreign key, the error code returned did not indicate that this was the case.
- `InnoDB`: Dropping a table where an `INSERT` was waiting for a lock to check a `FOREIGN KEY` constraint caused an assertion.
- `InnoDB`: The storgae of an SQL `NULL` value in some rare cases took more space than should have been required.
- Corrected a problem with `mysqld_safe` not properly capturing output from `ps`. (Bug #5878)
- `--expire-logs-days` [411] was not honored if using only transactions. (Bug #7236)
- Microseconds were dropped from the string result of the `STR_TO_DATE` function, when there was some other specifier in the format string following `%f`. (Bug #7458)
- `InnoDB`: Use the `fcntl(F_FULLFSYNC)` flush method on Mac OS X versions 10.3 and up instead of `fsync()` that could cause corruption at power outages.
- Added a `--default-character-set` option to `mysqladmin` to avoid problems when the default character set is not `latin1`. (Bug #7524)
- Linking both the MySQL client library and IMAP library in the same build failed. (Bug #7428)

- InnoDB: When innodb_file_per_table [1068] was enabled in my.cnf, records could disappear from the secondary indexes of a table after mysqld was killed.

  Note

  This fix introduced a new Bug #8021, affecting Windows and users of innodb_file_per_table [1068] only. If you are using innodb_file_per_table [1068] on Windows, you can work around this new issue by adding the line innodb_flush_method= unbuffered to the my.cnf or my.ini file.

  (Bug #7496)

- InnoDB: 32-bit mysqld binaries built on HP-UX 11 did not work with InnoDB files greater than 2 GB in size. (Bug #6189)

C.1.18 Changes in MySQL 4.1.8 (2004-12-14)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

Due to a libtool-related bug in the source distribution, the creation of shared libmysqlclient libraries was not possible (the resulting files were missing the .so file name extension). The file ltmain.sh was updated to fix this problem and the resulting source distribution was released as mysql-4.1.8a.tar.gz. This modification did not affect the binary packages. (Bug #7401)

Functionality Added or Changed

- **MySQL Cluster:** Added support for a [mysql_cluster] section to the my.cnf file for configuration settings specific to MySQL Cluster. The ndb-connectstring variable was moved here.
- **Replication:** mysqldump --single-transaction --master-data is now able to take an online (nonblocking) dump of InnoDB and report the corresponding binary log coordinates, which makes a backup suitable for point-in-time recovery, roll-forward or replication slave creation. See Section 4.5.4, “mysqldump — A Database Backup Program”.
- Added mysql_hex_string() C API function that hex-encodes a string.
- InnoDB: Do not periodically write SHOW INNODB STATUS information to a temporary file unless the configuration option innodb-status-file = 1 [1066] is set.
- FULLTEXT index block size is changed to be 1024 instead of 2048.
- InnoDB: Commit after every 10,000 copied rows when executing ALTER TABLE. This makes it much faster to recover from an aborted ALTER TABLE or OPTIMIZE TABLE.
- Added --order-by-primary [297] to mysqldump, to sort each table's data in a dump file. This may be useful when dumping a MyISAM table which will be loaded into an InnoDB table. Dumping a MyISAM table with this option is considerably slower than without.
- Added --hex-blob [295] option to mysqldump for dumping binary string columns using hexadecimal notation.

- The `--master-data` [296] option for `mysqldump` now takes an optional argument of 1 or 2 to produce a noncommented or commented `CHANGE MASTER TO` statement. The default is 1 for backward compatibility.
- The statements `CREATE TABLE`, `TRUNCATE TABLE`, `DROP DATABASE`, and `CREATE DATABASE` cause an implicit commit.
- Added `WITH CONSISTENT SNAPSHOT` clause to `START TRANSACTION` to begin a transaction with a consistent read.
- For `ALTER DATABASE`, the database name now can be omitted to apply the change to the default database.
- Automatic character set conversion formerly was done for operations that mix a column and a string such as assigning a string to a column, when this was possible without loss of information. Automatic conversion for operations that mix columns and strings has been expanded to cover many functions (such as `CONCAT()` [795]) and assignment operators. This reduces the frequency of `Illegal mix of collations` errors.
- Added `--disable-log-bin` [340] option to `mysqlbinlog`. Using this option you can disable binary logging for the statements produced by `mysqlbinlog`. That is, `mysqlbinlog --disable-log-bin <file_name> | mysql` won't write any statements to the MySQL server binary log.
- `mysqlbinlog` now prints an informative commented line (thread id, timestamp, server id, and so forth) before each `LOAD DATA INFILE`, like it does for other queries; unless `--short-form` is used.
- In the normal log MySQL now prints the log position for `Binlog Dump` requests.
- Added `--lock-all-tables` [295] to `mysqldump` to lock all tables by acquiring a global read lock.
- A connection doing a rollback now displays "Rolling back" in the `State` column of `SHOW PROCESSLIST`.

Bugs Fixed

- **Replication:** `OPTIMIZE TABLE`, `REPAIR TABLE`, and `ANALYZE TABLE` are now replicated without any error code in the binary log. (Bug #5551)
- **Replication:** `LOAD DATA INFILE` now works with option replicate-rewrite-db. (Bug #6353)
- **Replication:** Changed semantics of `CREATE/ALTER/DROP DATABASE` statements so that replication of `CREATE DATABASE` is possible when using `--binlog-do-db` [1182] and `--binlog-ignore-db` [1182]. (Bug #6391)
- **Replication:** `InnoDB`: If one used `INSERT IGNORE` to insert several rows at a time, and the first inserts were ignored because of a duplicate key collision, then `InnoDB` in a replication slave assigned `AUTO_INCREMENT` values 1 bigger than in the master. This broke the MySQL replication. (Bug #6287)
- Using a modified client library, a malicious user could take advantage of an issue in MySQL authentication code to crash the server with specially crafted packets. (Bug #7187)
- Prevent adding `CREATE TABLE .. SELECT` query to the binary log when the insertion of new records partially failed. (Bug #6682)
- Server warnings now are reset when you execute a prepared statement.
- `InnoDB`: Do not intentionally crash `mysqld` if the buffer pool is exhausted by the lock table; return error 1206 instead. Do not intentionally crash `mysqld` if we cannot allocate the memory for the `InnoDB` buffer pool. (Bug #6817, Bug #6827)

- If a connection was interrupted by a network error and did a rollback, the network error code got stored into the `BEGIN` and `ROLLBACK` binary log events; that caused superfluous slave stops. (Bug #6522)
- `InnoDB`: `innodb_data_file_path` [1067] was not handled correctly in some cases. This bug was introduced in MySQL 4.1.1.
- `InnoDB`: Let the `InnoDB` `FOREIGN KEY` parser remove the `latin1` character `0xA0` from the end of an unquoted identifier. The EMS MySQL Manager in `ALTER TABLE` adds that character after a table name, which caused error 121 when we tried to add a new constraint.
- A spurious "duplicate key" error resulted from executing a `REPLACE` or `INSERT ... ON DUPLICATE KEY UPDATE` statement performing a multiple-row insert on a table having unique and full-text indexes. (Bug #6784)
- `InnoDB`: Made the foreign key parser better aware of quotation marks. (Bug #6340)
- `mysqlbinlog` did not print `SET PSEUDO_THREAD_ID` statements in front of `LOAD DATA INFILE` statements inserting into temporary tables, thus causing potential problems when rolling forward these statements after restoring a backup. (Bug #6671)
- A reference to a column by name from a `WHERE` subquery to an outer query, with use of a temporary table by the outer query. (Bug #7079)
- A spurious `Record has changed since last read in table` error could be raised by some queries on `HEAP` tables containing only one row. (Bug #6748)
- Improved performance of identifier comparisons (if many tables or columns are specified).
- Execution of subqueries in `SET` and `DO` statements caused wrong results to be returned from subsequent queries. (Bug #6462)
- A multiple-table `DELETE` could cause MySQL to crash when using `InnoDB` tables. (Bug #6378, Bug #5837)
- If a connection had an open transaction but had done no updates to transactional tables (for example if had just done a `SELECT FOR UPDATE` then executed a nontransactional update, that update automatically committed the transaction (thus releasing `InnoDB`'s row-level locks etc). (Bug #5714)
- `INSERT` on a table with `FULLTEXT` indexes, could under rare circumstances result in a corrupted table if words of different lengths could be considered equal. This is possible in some collations such as `utf8_general_ci` and `latin1_german2_ci`. (Bug #6265)
- `mysqld_safe` was in many cases ignoring any `--no-defaults` [235], `--defaults-file` [235], or `--defaults-extra-file` [235] arguments. Those arguments are now honored, and this may change what options are passed to `mysqld` in some installations.
- A prepared statement using `SELECT ... PROCEDURE` could cause the server to crash.
- A prepared statement using a subquery could cause the server to crash.
- Starting and stopping the slave thread (only) could in some circumstance cause the server to crash. (Bug #6148)
- `NULL` was not always processed correctly in subqueries using `ALL` or `SOME`. (Bug #6247)
- MySQL required explicit privileges on system time zone description tables for implicit access to them (that is, if one set the `time_zone` [431] variable or used the `CONVERT_TZ()` [828] function) in cases where some table-level or column-level privileges already existed. (Bug #6765)

- `mysql_stmt_data_seek(stmt,0)` now rewinds a counter and enables buffered rows to be re-fetched on the client side. (Bug #6996)
- `NULL` were not handled caorrectly in cases of empty results in subqueries. (Bug #6806)
- Some internal structures were not initialized correctly prior to first execution. (Bug #6517)
- `InnoDB`: `innodb_locks_unsafe_for_binlog` [1070] still uses next-key locking, which is unnecessary next-key. Such locks are now removed when the `innodb_locks_unsafe_for_binlog` [1070] option is enabled. (Bug #6747)
- Some complex queries did not work correctly with subqueries. (Bug #6841, Bug #6406)
- If `STMT_ATTR_UPDATE_MAX_LENGTH` is set for a prepared statement, `mysql_stmt_store_result()` updates `field->max_length` for numeric columns as well. (Bug #6096)
- `InnoDB`: `FOREIGN KEY` constraints treated table and database names as case-insensitive, so that `RENAME TABLE t TO T` would hang in an endless loop if `t` had a foreign key constraint defined on it. The server would also hang if one tried using an `ALTER TABLE` or `RENAME TABLE` statement to create a foreign key constraint name that collided with existing one. (Bug #3478)
- A prepared statement using `SELECT * FROM t1 NATURAL JOIN t2...` could cause the server to crash.
- `InnoDB`: Do not call `rewind()` when displaying `SHOW INNODB STATUS` information on `stderr`.
- Using the string function `LEFT` as part of the expression used as `GROUP BY` column caused the server to crash. (Bug #7101)
- The server was interpreting `CHAR BINARY` and `VARCHAR BINARY` columns from 4.0 tables as having the `BINARY` and `VARBINARY` data types. Now they are interpreted as `CHAR` and `VARCHAR` columns that have the binary collation of the column's character set. (This is the same way that `CHAR BINARY` and `VARCHAR BINARY` are handled for new tables created in 4.1.)
- `GROUP_CONCAT(...ORDER BY)` [883] when used with prepared statements gave wrong sorting order.
- `INSERT ... SELECT` no longer reports spurious "column truncated" warnings (Bug #6284)
- The server accepted datetime values with an invalid year part. The server now also performs the same checks for datetime values passed through `MYSQL_TIME` structures as for datetime values passed as strings. (Bug #6266)
- Prepared statements now handle `ZEROFILL` when converting `integer` to `string`.
- `CREATE TABLE created_table` didn't signal when table was created. This could cause a `DROP TABLE created_table` in another thread to wait "forever".
- A sequence of `BEGIN` (or `SET autocommit = 0`), `FLUSH TABLES WITH READ LOCK`, transactional update, `COMMIT`, `FLUSH TABLES WITH READ LOCK` could hang the connection forever and possibly the MySQL server itself. This happened for example when running the `innobackup` script several times. (Bug #6732)
- A rare memory corruption problem could cause `MATCH ... AGAINST` on columns using multi-byte character sets to crash the server. (Bug #6269)
- A call to `mysql_stmt_store_result()` occurred without a preceding call to `mysql_stmt_bind_result()` caused the server to crash.

- Insufficient privilege checks were made for `SHOW CREATE TABLE`. (Bug #7043)
- `InnoDB`: Refuse to open new-style tables created with MySQL 5.0.3 or later. (Bug #7089)
- Backported a fix for the full-text interface from MySQL 5.0. (Bug #6523)

C.1.19 Changes in MySQL 4.1.7 (2004-10-23, Production)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- Added a startup option and settable system variable `innodb_table_locks` [1072] for making `LOCK TABLE` acquire locks on `InnoDB` tables. The default value is 1, which means that `LOCK TABLES` also causes `InnoDB` to take a table lock internally. In applications using `autocommit = 1` [406] and `LOCK TABLES`, `InnoDB`'s internal table locks (added in MySQL 4.0.20 and 4.1.2) can cause deadlocks. You can set `innodb_table_locks = 0` in `my.cnf` to remove that problem.

  In addition, `SHOW TABLE STATUS` now shows the creation time of `InnoDB` tables. That this timestamp might not always be correct because (for example) it was changed by `ALTER TABLE`. See Section 13.2.15, “Restrictions on `InnoDB` Tables”. (Bug #3299, Bug #5998)
- `InnoDB`: If `innodb_thread_concurrency` [1072] would be exceeded, let a thread sleep 10 ms before entering the FIFO queue; previously, the value was 50 ms.
- `MOD()` [821] no longer rounds arguments with a fractional part to integers. Now it returns exact remainder after division. (Bug #6138)

Bugs Fixed

- **Security Fix:** A missing `UPDATE` [493] privilege could be circumvented by a user having `INSERT` [492] and `SELECT` [492] privileges for table with a primary key. (Bug #6173)
- **Replication:** A problem introduced in MySQL 4.0.21 caused replication slaves to stop (error 1223) where a connection started a transaction, performed updates, then issued a `FLUSH TABLES WITH READ LOCK` followed by a `COMMIT`. This issue occurred when using the `InnoDB` `innobackup` script. (Bug #5949)
- `DATE`, `TIME`, and `DATETIME` columns were not handled correctly by the binary protocol. The problem was compiler-specific and could have been observed on HP-UX, AIX, and Solaris 9, when using native compilers. (Bug #6025)
- Selecting from a `HEAP` table with `key_column IS NOT NULL` could cause the server to crash. The crash could also occur even if all index parts were not used. (Bug #6082)
- `FOUND_ROWS()` [872] did not work correctly with `LIMIT` clause in prepared statements. (Bug #6088)
- `libmysqlclient` did not convert zero date values (`0000-00-00`) to strings correctly. (Bug #6058)
- Invoking the deprecated `libmysqlclient` function `mysql_create_db()` caused the server to crash. (Bug #6081)
- `MyISAM` indexes could be corrupted when key values started with character codes below `BLANK`. This was caused by the new key sort order instroduced in MySQL 4.1. (Bug #6151)

- InnoDB: Release the dictionary latch during a long cascaded FOREIGN KEY operation, so that we do not starve other users doing CREATE TABLE or other DDL operation. This caused a notorious 'Long semaphore wait' message to be printed to the .err log. (Bug #5961)
- Now implicit access to system time zone description tables (which happens when you set the time_zone [431] variable or use CONVERT_TZ() [828] function) does not require any privileges. (Bug #6116)
- TINYINT columns were not handled correctly in the binary protocol. The problem was specific to platforms where the C compiler has the char data type unsigned by default. (Bug #6024)
- libmysqlclient did not convert negative time values to strings correctly. (Bug #6049)
- InnoDB: LOAD DATA INFILE…REPLACE printed duplicate key errors when executing the same LOAD statement several times. (Bug #5835)
- Attempting to prepare a statement with RAND(?) [822] caused the server to crash. (Bug #5985)
- Bad metadata was sent for SELECT statements not returning a result set (such as SELECT ... INTO OUTFILE) by the prepared statements protocol. (Bug #6059)
- NATURAL JOIN did not work correctly in prepared statements. . (Bug #6046)
- REVOKE ALL PRIVILEGES, GRANT OPTION FROM *user* did not remove all privileges correctly. (Bug #5831)
- Join of tables from different databases having columns with identical names did not work correctly, returning the error Column 'xxx' in field list is ambiguous. (Bug #6050)

C.1.20 Changes in MySQL 4.1.6 (2004-10-10)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- Now if ALTER TABLE converts one AUTO_INCREMENT column to another AUTO_INCREMENT column it preserves zero values (this includes the case that we don't change such column at all).
- On Windows, the MySQL configuration files included in the package now use .ini instead of .cnf as the file name suffix.
- If a write to a MyISAM table fails because of a full disk or an exceeded disk quota, it now prints a message to the error log every 10 minutes, and waits until disk space becomes available. (Bug #3248)
- InnoDB: The innodb_autoextend_increment [1067] startup option that was introduced in release 4.1.5 was made a settable global variable. (Bug #5736)
- InnoDB: If DROP TABLE is invoked on an InnoDB table for which the .ibd file is missing, print to error log that the table was removed from the InnoDB data dictionary, and enable MySQL to delete the .frm file. Maybe DROP TABLE should issue a warning in this case.
- InnoDB: Added the startup option and settable global variable innodb_max_purge_lag [1071] for delaying INSERT, UPDATE and DELETE operations when the purge operations are lagging. The default value of this parameter is zero, meaning that there are no delays. See Section 13.2.10, “InnoDB Multi-Versioning”.

- Added option `--sigint-ignore` to the `mysql` command line client to make it ignore `SIGINT` signals (typically the result of the user pressing **Control+C**).
- Now if `ALTER TABLE` converts some column to `TIMESTAMP NOT NULL` column it converts `NULL` values to current timestamp value (One can still get old behavior by setting system `TIMESTAMP` variable to zero).
- `TIMESTAMP` columns now can store `NULL` values. To create such a column, you must explicitly specify the `NULL` attribute in the column specification. (Unlike all other data types, `TIMESTAMP` columns are `NOT NULL` by default.)

Bugs Fixed

- **Replication:** `SET COLLATION_SERVER...` statements replicated by the slave SQL thread no longer advance its position. This is so that, if the thread is interrupted before the update is completed, it later performs the `SET` again. (Bug #5705)
- The server sometimes chose a nonoptimal execution plan for a prepared statement executed with changed placeholder values. (Bug #6042)
- Behavior of `ALTER TABLE` converting column containing `NULL` values to `AUTO_INCREMENT` column is no longer affected by `NO_AUTO_VALUE_ON_ZERO` [458] mode. . (Bug #5915)
- `InnoDB`: `CREATE TEMPORARY TABLE ... ENGINE=InnoDB` terminated `mysqld` when running in `innodb_file_per_table` [1068] mode. Now, per-table for temporary tables are created in the temporary directory used by `mysqld`. (Bug #5137)
- `InnoDB`: The `FOREIGN KEY` parser did not permit `ALTER TABLE` on tables whose names contained `#` characters. (Bug #5856)
- `InnoDB`: `ALTER TABLE t DISCARD TABLESPACE` did not work correctly. (Bug #5851)
- `InnoDB`: Change error code to `HA_ERR_ROW_IS_REFERENCED` if we cannot `DROP` a parent table referenced by a `FOREIGN KEY` constraint; this error number is less misleading than the previous number `HA_ERR_CANNOT_ADD_FOREIGN`, but misleading still. (Bug #6202)
- An attempt to execute a prepared statement with a subquery inside a boolean expression caused the server to crash. (Bug #5987)
- The server crashed when character set conversion was implicitly used in prepared mode, as in `'abc' LIKE CONVERT('abc' as utf8)`. (Bug #5688)
- The `mysql_change_user()` C API function now frees all prepared statements associated with the connection. (Bug #5315)
- `InnoDB`: `SHOW CREATE TABLE` now obeys the `SET sql_mode = ANSI` and `SET sql_quote_show_create = 0` settings. (Bug #5292)
- `InnoDB`: If one updated a column so that its size changed, or updated it to an externally stored (`TEXT` or `BLOB`) value, then ANOTHER externally stored column would show up as 512 bytes of good data + 20 bytes of garbage in a consistent read that fetched the old version of the row. (Bug #5960)
- `InnoDB`: UTF-8 characters were not always handled correctly in column prefix indexes. (Bug #5975)
- If the slave SQL thread finds a syntax error in a query (which should be rare, as the master parsed it successfully), it now stops immediately. (Bug #5711)
- Inserting `NULL` into an `AUTO_INCREMENT` column failed when using prepared statements. (Bug #5510)

- InnoDB: Make the check for excessive semaphore waits tolerate glitches in the system clock (do not crash the server if the system time is adjusted while InnoDB is under load.). (Bug #5898)

C.1.21 Changes in MySQL 4.1.5 (2004-09-16)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- InnoDB: Added the configuration option `innodb_autoextend_increment` [1067] for setting the size in megabytes by which InnoDB tablespaces are extended when they become full. The default value is 8, corresponding to the fixed increment of 8MB in previous versions of MySQL.
- InnoDB: The new Windows installation wizard of MySQL makes InnoDB as the MySQL default table type on Windows, unless explicitly specified otherwise. Note that it places the `my.ini` file in the installation directory of the MySQL server. See Section 2.3.4.14, “The Location of the my.ini File”.

Bugs Fixed

- Providing '0000-00-00' date as a prepared statement parameter value led to a server crash. (Bug #4231, Bug #4562)
- Detection of using the same table for updating and selecting in multi-update queries was not done correctly. (Bug #5455)
- The internal field length of integer user variables was incorrect. This showed up when creating a table as `SELECT @var_name`. (Bug #4788)
- After reaching a certain limit of prepared statements per connection (97), statement IDs began to overlap, so occasionally wrong statements were chosen for execution. (Bug #5399)
- The syntax analyzer did not handle the `IGNORE_SPACE` [458] server SQL mode correctly, using (for example) `default.07` in place of `default .07`. (Bug #5318)
- `OPTIMIZE TABLE` could cause table corruption on `FULLTEXT` indexes. (Bug #5327)
- Executing a prepared statement with `BETWEEN ? AND ?` [783] and a datetime column caused the server to crash. (Bug #5748)
- Executing a statement containing thousands of placeholders caused a buffer overflow in the prepared statements API (`libmysqlclient`). (Bug #5194)
- Name resolution of external columns of subqueries was done correctly if the subquery was placed in the select list of the outer query and used grouping. (Bug #5326)
- A prepared statement using `LIKE` and called with arguments in different character sets caused the server to crash. (Bug #4368)
- InnoDB: A maximum of 1000 connections could occur inside InnoDB at the same time, a higher number causing an assertion failure. Now the maximum can be much higher, and depends on the buffer pool size. (Bug #5414)
- The values of the `max_sort_length` [420], `sql_mode` [429], and `group_concat_max_len` [412] system variables now are stored in the query cache with other query information to avoid returning an incorrect result from the query cache. (Bug #5394, Bug #5515)

C.1.22 Changes in MySQL 4.1.4 (2004-08-26, Gamma)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

To fix a compile problem on systems that do not have `automake` 1.7 installed, an updated 4.1.4a source tarball has been published. In addition to resolving this `automake` dependency (Bug #5319), it also fixes some reported `libedit` compile errors when using a non-`gcc` compiler (Bug #5353).

Functionality Added or Changed

- Made internal representation of `TIMESTAMP` values in `InnoDB` in 4.1 to be the same as in 4.0. This difference resulted in incorrect datetime values in `TIMESTAMP` columns in `InnoDB` tables after an upgrade from 4.0 to 4.1. **Warning: extra steps during upgrade required!** Unfortunately this means that if you are upgrading from 4.1.x, where x <= 3, to 4.1.4 you should use `mysqldump` for saving and then restoring your `InnoDB` tables with `TIMESTAMP` columns. (Bug #4492)
- The `mysqld-opt` Windows server was renamed to `mysqld`. This completes the Windows server renaming begun in MySQL 4.1.2. See Section 2.3.8, “Selecting a MySQL Server Type”.
- Added `--start-datetime` [341], `--stop-datetime` [342], `--start-position` [342], and `--stop-position` [342] options to `mysqlbinlog`. These make point-in-time recovery easier.
- Killing a `CHECK TABLE` statement does not result in the table being marked as “corrupted” any more; the table remains as if `CHECK TABLE` had not even started. See Section 12.4.6.3, “`KILL` Syntax”.
- Added the `CSV` storage engine.
- Added Latin language collations for the `ucs2` and `utf8` Unicode character sets. These are called `ucs2_roman_ci` and `utf8_roman_ci`.
- Corrected the name of the Mac OS X StartupItem script (it must match the name of the subdirectory, which was renamed to `MySQLCOM` in MySQL 4.1.2). Thanks to Bryan McCormack for reporting this.
- Made the MySQL server ignore `SIGHUP` and `SIGQUIT` on Mac OS X 10.3. This is needed because under this OS, the MySQL server receives lots of these signals. (Bug #2030)
- Support of usage of column aliases qualified by table name or alias in `ORDER BY` and `GROUP BY` was dropped. For example the following query `SELECT a AS b FROM t1 ORDER BY t1.b` is not permitted. One should use `SELECT a AS b FROM t1 ORDER BY t1.a` or `SELECT a AS b FROM t1 ORDER BY b` instead. This was nonstandard (since aliases are defined on query level not on table level) and caused problems with some queries.

Bugs Fixed

- **Replication:** When the slave SQL thread was replicating a `LOAD DATA INFILE` statement, it didn't show the statement in the output of `SHOW PROCESSLIST`. (Bug #4326)
- **Replication:** When a multiple-table `DROP TABLE` failed to drop a table on the master server, the error was not written to the binary log. (Bug #4553)
- **Replication:** A `CREATE TABLE ... TYPE=HEAP ... AS SELECT...` statement caused the replication slave to stop. (Bug #4971)

- `libmysqlclient` crashed when attempting to fetch the value of a `MEDIUMINT` column. (Bug #5126)
- Executing `UNHEX(NULL)` [803] caused the server to crash. (Bug #4441)
- The `.err` extension was omitted from the error log file (`--log-error` [388]) when the host name contained a domain name. The domain name is now replaced by the extension. (Bug #4997)
- `myisasmchk --extend-check` [316] crashed when run on a list of files. (Bug #4901)
- `disable-local-infile` option had no effect if the client read it from a configuration file using `mysql_options(...,MYSQL_READ_DEFAULT,...)`. (Bug #5073)
- The MySQL server crashed when attempting to execute a prepared statement with `SELECT ... INTO @var` for the second time. (Bug #5034)
- `mysqlbinlog --position --read-from-remote-server` had incorrect output for `# at log_pos`. (Bug #4506)
- An error was reported when a column from an `ORDER BY` clause was present in two tables participating in a `SELECT`, even if the second instance of column in select list was renamed. (For example, `SELECT t1.a AS c FROM t1, t2 ORDER BY a` produced an error if both `t1` and `t2` tables contain column `a`).

  Now MySQL does not prefer columns, mentioned in a select list but renamed, over columns from other tables participating in a `FROM` clause when it resolves the `ORDER BY` clause. (Bug #4302)
- `FLUSH TABLES WITH READ LOCK` now blocks `COMMIT` statements if the server is running with binary logging enabled; this ensures that the binary log position is trustable when doing a full backup of tables and the binary log. (Bug #4953)
- Concurrent accesses to more than one `MERGE` table, or to one `MERGE` table and a `MyISAM` tables, could result in a crash or hang of the server. (Bug #2408)
- `mysql-test-run` failed the `rpl_trunc_binlog` test when running the test from the installation directory. (Bug #5050)
- A deadlock could happen under certain rare circumstances when using `KILL`. (Bug #4810)
- Support for `%T, %r, %V, %v` and `%X, %x` format specifiers was added to `STR_TO_DATE()` [838] function. (Bug #4756)
- `MATCH ... AGAINST` now works in a subquery. (Bug #4769)
- `mysql-test-run` failed the `grant_cache` test when run as Unix root user. (Bug #4678)
- Prohibited resolving of table fields in inner queries if fields do not take part in grouping for queries with grouping (inside aggregate function arguments, all table fields are still permitted). (Bug #4814)
- `SET GLOBAL SYNC_BINLOG` did not work on some platforms (Mac OS X). (Bug #5064)
- If `CREATE TEMPORARY TABLE t SELECT` failed while loading the data, the temporary table was not dropped. (Bug #4551)
- An assertion failure could occur when reading the grant tables (Bug #4407)
- Using the `CONVERT_TZ()` [828] function with a time zone described in the database as parameter where this time zone had not been used before caused the server to crash. (Bug #4508)
- Fixed a crash after `STOP SLAVE` if the IO replication thread is in the state `Waiting to reconnect after a failed master event read`. (Bug #4629)

- `NATURAL JOIN` where the joined tables had no common column caused the server to hang. (Bug #4807)
- `mysql_fix_privilege_tables` did not handle the `--password=password_val` option correctly. (Bug #4240, Bug #4543)
- `mysqlbinlog --read-from-remote-server` sometimes could not accept 2 binary logs in a single invocation. (Bug #4507)
- The counter for an `AUTO_INCREMENT` column was not reset by `TRUNCATE TABLE` if the table was a temporary one. (Bug #5033)
- `KILL`ing a connection while it was performing `START SLAVE` caused the server to crash. (Bug #4827)
- Execution `IN` subqueries that use compound indexes was better optimized. (Bug #4435)
- `mysql_options(...,MYSQL_OPT_LOCAL_INFILE,...)` failed to disable `LOAD DATA LOCAL INFILE`. (Bug #5038)
- Attempting to execute for a second time a prepared statement with `NOT` in an `WHERE` or `ON` clause caused the server to crash. (Bug #4912)

C.1.23 Changes in MySQL 4.1.3 (2004-06-28, Beta)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

The initial release of MySQL 4.1.3 for Windows was accidentally compiled without support for the Spatial Extensions (OpenGIS). This was fixed by rebuilding from the same 4.1 code snapshot with the missing option and releasing those packages as version 4.1.3a.

To enable compiling the newly released PHP 5 against MySQL 4.1.3 on Windows, the Windows packages had to be rebuilt once more to add a few missing symbols to the MySQL client library. These packages were released as MySQL 4.1.3b.

Functionality Added or Changed

- **Incompatible Change:** The `timezone` [432] system variable has been removed and replaced by `system_time_zone` [430]. See Section 5.1.3, “Server System Variables”.
- **Incompatible Change:** C API change: `mysql_shutdown()` now requires a second argument. This is a source-level incompatibility that affects how you compile client programs; it does not affect the ability of compiled clients to communicate with older servers. See Section 17.6.6.63, “`mysql_shutdown()`”.
- **Replication:** Replication and `mysqlbinlog` now have better support for the case that the session character set and collation variables are changed within a given session. See Section 14.7, “Replication Features and Issues”.
- **Replication:** `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, single-table `DELETE` and single-table `UPDATE` are now written to the binary log even if they changed nothing on the master (for example, even if the `DELETE` matched no row). The old behavior sometimes caused bad surprises in replication setups.

- Added SQL syntax for prepared statements. See Section 12.6, “SQL Syntax for Prepared Statements”.
- Added the `sync_binlog=N` global variable and startup option, which makes the MySQL server synchronize its binary log to disk (`fdatasync()`) after every *N*th write to the binary log.
- `CHECK TABLE` now can be killed. It then marks the table as corrupted. See Section 12.4.6.3, “`KILL` Syntax”.
- `OPTIMIZE TABLE` for `InnoDB` tables is now mapped to `ALTER TABLE` instead of `ANALYZE TABLE`.
- Added the `ARCHIVE` storage engine.
- Added `--innodb-safe-binlog` [387] server option, which adds consistency guarantees between the content of `InnoDB` tables and the binary log. See Section 5.3.4, “The Binary Log”.
- `sync_frm` [430] is now a settable global variable (not only a startup option).
- Support for per-connection time zones was added. Now you can set the current time zone for a connection by setting the `@@time_zone` system variable to a value such as `'+10:00'` or `'Europe/Moscow'` (where `'Europe/Moscow'` is the name of one of the time zones described in the system tables). Functions like `CURRENT_TIMESTAMP`, `UNIX_TIMESTAMP`, and so forth honor this time zone. Values of `TIMESTAMP` type are also interpreted as values in this time zone. So now our `TIMESTAMP` type behaves similar to Oracle's `TIMESTAMP WITH LOCAL TIME ZONE`. That is, values stored in such a column are normalized toward UTC and converted back to the current connection time zone when they are retrieved from such a column. To set up the tables that store time zone information, see Section 2.10, “Postinstallation Setup and Testing”.
- `LIKE` now supports the use of a prepared statement parameter or delimited constant expression as the argument to `ESCAPE` . (Bug #4200)
- Language-specific collations were added for the `ucs2` and `utf8` Unicode character sets: Icelandic, Latvian, Romanian, Slovenian, Polish, Estonian, Swedish, Turkish, Czech, Danish, Lithuanian, Slovak, Spanish, Traditional Spanish.
- Changed the slave SQL thread to print fewer useless error messages (no more message duplication; no more messages when an error is skipped (because of `slave-skip-errors`).
- Basic time zone conversion function `CONVERT_TZ()` [828] was added. It assumes that its first argument is a datetime value in the time zone specified by its second argument and returns the equivalent datetime value in the time zone specified by its third argument.

Bugs Fixed

- **Replication:** Complex expressions using `AND`, `OR`, or both could result in a crash if the query containing the expression query was ignored, either by a replication server due to `--replicate-*-table` rules, or by any MySQL server due to a syntax error. (Bug #3969, Bug #4494)
- **Replication:** The slave SQL thread refused to replicate `INSERT ... SELECT` if it examined more than 4 billion rows. (Bug #3871)
- Re-execution of optimized `COUNT(*)` [882], `MAX()` [884], and `MIN()` [884] functions is now handled correctly for prepared statements. (Bug #2687)
- Different numbers of warnings were generated when an invalid datetime (as a string or as a number) was inserted into a `DATETIME` or `TIMESTAMP` column. (Bug #2336)
- During the installation process of the server RPM on Linux, if `mysqld` was run as the `root` system user and with `--log-bin` [1181] pointing to a directory outside of `/var/lib/mysql`, it created binary log

files owned by `root` in this directory, which remained owned by `root` after the installation. Now `mysqld` is started as the `mysql` system user instead. (Bug #4038)

- A potential memory overrun could occur in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017)
- `CREATE DATABASE IF NOT EXISTS` caused an error on Win32 platforms if the database existed. (Bug #4378)
- Attempt to prepare a statement containing a character set introducer caused the server to crash. (Bug #4105)
- Attempting to execute a nonprepared statement could cause the server to crash. (Bug #4236)
- `mysqlbinlog` didn't escape the string content of user variables, and did not deal well when these variables were in non-ASCII character sets; this is now fixed by always printing the string content of user variables in hexadecimal. The character set and collation of the string is now also printed. (Bug #3875)
- `UNION` returned incorrect results if the display length of columns for numeric types was set to less than the actual length of values in them. (Bug #4067)
- Conversion of a client-side string column to a `MYSQL_TIME` application buffer was not handled correctly by the prepared statements API. (Bug #4030)
- Prepared `EXPLAIN` statements could lead to a server crash. (Bug #4271)
- A malicious user could bypass password verification with specially crafted packets, using a modified client library. (CVE-2004-0627, CVE-2004-0628)
- `NULL` was not handled correctly with derived tables. (Bug #4097)
- `MERGE` tables created with `INSERT_METHOD=LAST` were not able to report a key number, causing `Duplicate entry` errors for `UNIQUE` keys in `INSERT` statements. As a result, the error message was not precise enough (error 1022 instead of error 1062) and `INSERT ... ON DUPLICATE KEY UPDATE` did not work. (Bug #4008)
- Parameters in some prepared statements were not handled correctly. (Bug #4280)
- Prepared statements did not always work correctly on big-endian platforms. (Bug #4173)
- `CONCAT(?, col_name)` [795], when used in prepared statements, returned incorrect results. (Bug #3796)
- Tables were unlocked too early in cases of a subquery in a query's `HAVING` clause. (Bug #3984)
- The range optimizer did not perform correctly when using many `IN()` queries on different key parts. (Bug #4157)
- Performance of `COUNT(DISTINCT)` [882] degraded in cases like `COUNT(DISTINCT a TEXT, b CHAR(1))` [882] (no index used). (Bug #3904)
- In rare circumstances, `DELETE` from a table with `FULLTEXT` indexes resulted in a corrupted table, if words of different lengths could be considered equal. This is possible with some collations, for example, `utf8_general_ci` and `latin1_german2_ci`. (Bug #3808)
- `mysql_stmt_close()` hung when attempting to close a statement after failed `mysql_stmt_fetch()` call. (Bug #4079)
- Using `--with-charset` [97] with `configure` did not affect the MySQL client library. (Bug #3990)

- Added missing `root` account to Windows version of `mysqld`. (Bug #4242)
- The microseconds part of `MYSQL_TYPE_TIME`/`MYSQL_TYPE_DATETIME` columns was not sent to the client by prepared statements. (Bug #4026)
- `mysqldump` when it did not return any error if the output device was full. (Bug #1851)
- Made `DROP DATABASE` honor the value of `lower_case_table_names` [418]. (Bug #4066)
- Under rare circumstances, `MATCH ... AGAINST(... IN BOOLEAN MODE)` could yield incorrect results if, in the collation used for the data, one byte could match many (as in `utf8_general_ci` and `latin1_german2_ci`.) (Bug #3964)

C.1.24 Changes in MySQL 4.1.2 (2004-05-28)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- **Security Fix:** The `--defaults-file=file_name` [235] option now requires that the file name must exist. (Bug #3413)
- **Incompatible Change:** The signature of the `mysql_stmt_prepare()` function was changed to `int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)`. To create a `MYSQL_STMT` handle, you should use the `mysql_stmt_init()` function, not `mysql_stmt_prepare()`.
- **Incompatible Change:** Renamed prepared statements C API functions:

Old Name	New Name
`mysql_bind_param()`	`mysql_stmt_bind_param()`
`mysql_bind_result()`	`mysql_stmt_bind_result()`
`mysql_prepare()`	`mysql_stmt_prepare()`
`mysql_execute()`	`mysql_stmt_execute()`
`mysql_fetch()`	`mysql_stmt_fetch()`
`mysql_fetch_column()`	`mysql_stmt_fetch_column()`
`mysql_param_count()`	`mysql_stmt_param_count()`
`mysql_param_result()`	`mysql_stmt_param_metadata()`
`mysql_get_metadata()`	`mysql_stmt_result_metadata()`
`mysql_send_long_data()`	`mysql_stmt_send_long_data()`

  Now all functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.
- **Incompatible Change:** Added support for `DEFAULT CURRENT_TIMESTAMP` and for `ON UPDATE CURRENT_TIMESTAMP` specifications for `TIMESTAMP` columns. Now you can explicitly say that a `TIMESTAMP` column should be set automatically to the current timestamp for `INSERT` or `UPDATE` statements, or even prevent the column from updating automatically. Only one column with such an auto-set feature per table is supported. `TIMESTAMP` columns created with earlier versions of MySQL behave as before. Behavior of `TIMESTAMP` columns that were created without explicit specification of default/on as earlier depends on its position in table: If it is the first `TIMESTAMP` column, it be

treated as having been specified as `TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP`. In other cases, it would be treated as a `TIMESTAMP DEFAULT 0` column. `NOW` is supported as an alias for `CURRENT_TIMESTAMP`.

Unlike in previous versions, explicit specification of default values for `TIMESTAMP` column is never ignored and turns off the auto-set feature (unless you have `CURRENT_TIMESTAMP` as the default).

- **Incompatible Change:** String comparison now works according to the SQL standard. Because we have that `'a' = 'a '` then from it must follow that `'a' > 'a\t'`. (The latter was not the case before MySQL 4.1.2.) To implement it, we had to change how storage engines compare strings internally. As a side effect, if you have a table where a `CHAR` or `VARCHAR` column in some row has a value with the last character less than `ASCII(32)` [793], you have to repair this table. `CHECK TABLES` tells you if this problem exists. (Bug #3152)

- **Incompatible Change:** Handling of the `FLOAT` and `DOUBLE` floating-point data types is more strict to follow standard SQL. For example, a data type of `FLOAT(3,1)` stores a maximum value of 99.9. Previously, the server permitted larger numbers to be stored. That is, it stored a value such as 100.0 as 100.0. Now the server clips 100.0 to the maximum permissible value of 99.9. If you have tables that were created before MySQL 4.1.2 and that contain floating-point data not strictly legal for the column type, you should alter the data types of those columns. For example:

  ```
  ALTER TABLE tbl_name MODIFY col_name FLOAT(4,1);
  ```

- **Incompatible Change:** The `Type` output column for `SHOW TABLE STATUS` now is labeled `Engine`.

- **Replication:** For replication of `MEMORY` (`HEAP`) tables: Made the master automatically write a `DELETE FROM` statement to its binary log when a `MEMORY` table is opened for the first time since the master's startup. This is for the case where the slave has replicated a nonempty `MEMORY` table, and then the master is shut down and restarted: the table is now empty on the master; the `DELETE FROM` empties it on the slave as well.

  Even with this fix, between the master's restart and the first use of the table on master, the slave still has out-of-date data in the table. However, if you use the `--init-file` [387] option to populate the `MEMORY` table on the master at startup, it ensures that the failing time interval is zero. (Bug #2477)

- **Replication:** Added option `--replicate-same-server-id` [1177].

- **Replication:** `UUID()` [879] function implemented. Note that it does not work with replication yet. See Section 11.14, “Miscellaneous Functions”.

- MySQL now issues a warning when a `SET` or `ENUM` column with duplicate values in the list is created. (Bug #1427)

- The `ft_boolean_syntax` [411] variable now can be changed while the server is running. See Section 5.1.3, “Server System Variables”.

- `MyISAM` tables now support keys up to 1000 bytes long.

- `mysql` command-line client now supports multiple `-e` options. (Bug #591)

- `CHAR BYTE` is an alias for the `BINARY` data type. (Previously, it was an alias for `CHAR BINARY`.)

- `mysqlhotcopy` now works on NetWare.

- Prepared statements now work with all types of subqueries.

- Added the `--default-storage-engine` [386] server option as a synonym for `--default-table-type` [386].

- MyISAM and InnoDB tables now support index prefix lengths up to 1000 bytes long.
- The mysqld Windows server was renamed to mysqld-debug. See Section 2.3.8, “Selecting a MySQL Server Type”.
- The MySQL server now returns an error if SET sql_log_bin or SET sql_log_update is issued by a user without the SUPER [493] privilege (in previous versions it just silently ignored the statement in this case).
- Added init_connect [414] and init_slave [1180] system variables. The values should be SQL statements to be executed when each client connects or each time a slave's SQL thread starts, respectively.
- Added the mysql_set_local_infile_handler() and mysql_set_local_infile_default() C API functions.
- Added Handler_discover [1306] status variable.
- C API enhancement: SERVER_QUERY_NO_INDEX_USED and SERVER_QUERY_NO_GOOD_INDEX_USED flags are now set in the server_status field of the MYSQL structure. It is these flags that make the query to be logged as slow if mysqld was started with --log-slow-queries [388] --log-queries-not-using-indexes [388].
- The FLOAT and DECIMAL types now obey (precision,scale) settings. (Bug #10897)
- The mysql command-line client no longer stores in the history file multiple copies of identical queries that are run consecutively.
- If you try to create a key with a key part that is too long, and it is safe to auto-truncate it to a smaller length, MySQL now does so. A warning is generated, rather than an error.
- The improved character set support introduced in MySQL 4.1.0 for the MyISAM and HEAP storage engines is now available for InnoDB as well.
- Added latin1_spanish_ci (Modern Spanish) collation for the latin1 character set.
- Added Binlog_cache_use [449] and Binlog_cache_disk_use [449] status variables that count the number of transactions that used transaction binary log and that had to flush this temporary binary log to disk instead of using only the in-memory buffer. They can be used for tuning the binlog_cache_size [407] system variable.
- Added the storage_engine [430] system variable as a synonym for table_type [431].
- Added explanation of hidden SELECT of UNION in output of EXPLAIN SELECT statement.
- Added the EXAMPLE storage engine.
- Internal string-to-number conversion now supports only SQL:2003 compatible syntax for numbers. In particular, '0x10'+0 does not work anymore. (Actually, it worked only on some systems before, such as Linux. It did not work on others, such as FreeBSD or Solaris. Making these queries OS-independent was the goal of this change.) Use CONV() [818] to convert hexadecimal numbers to decimal. Example: CONV(MID('0x10',3),16,10)+0 [818].
- The --log-warnings [389] server option now is enabled by default. Disable with --log-warnings=0 [389].
- Added the ENGINE table option as a synonym for the TYPE option for CREATE TABLE and ALTER TABLE.

- `UNHEX()` [803] function implemented. See Section 11.5, “String Functions”.
- Multi-line statements in the `mysql` command-line client now are stored in the history file as a single line.
- `ALTER TABLE DROP PRIMARY KEY` no longer drops the first `UNIQUE` index if there is no primary index. (Bug #2361)
- The Mac OS X Startup Item has been moved from the directory `/Library/StartupItems/MySQL` to `/Library/StartupItems/MySQLCOM` to avoid a file name collision with the MySQL Startup Item installed with Mac OS X Server. See Section 2.12.2, “Mac OS X Notes”.
- MySQL now supports up to 64 indexes per table.
- New `myisam_data_pointer_size` [421] system variable. See Section 5.1.3, “Server System Variables”.
- A name of “Primary” no longer can be specified as an index name. (That name is reserved for the `PRIMARY KEY` if the table has one.) (Bug #856)
- Now `sql_select_limit` [430] variable has no influence on subqueries. (Bug #2600)
- `REVOKE ALL PRIVILEGES, GRANT FROM user_list` is changed to a more consistent `REVOKE ALL PRIVILEGES, GRANT OPTION FROM user_list`. (Bug #2642)
- Added option `--to-last-log` to `mysqlbinlog`, for use in conjunction with `--read-from-remote-server`.
- When a session having open temporary tables terminates, the statement automatically written to the binary log is now `DROP TEMPORARY TABLE IF EXISTS` instead of `DROP TEMPORARY TABLE`, for more robustness.
- Added support for character set conversion and `MYSQL_TYPE_BLOB` type code in prepared statement protocol.
- Changed that when the MySQL server has binary logging disabled (that is, no `--log-bin` [1181] option was used), then no transaction binary log cache is allocated for connections. This should save `binlog_cache_size` [407] bytes of memory (32KB by default) for every connection.
- `mysqld_multi` now creates the log in the directory named by `datadir` (from the `[mysqld]` section in `my.cnf` or compiled in), not in `/tmp`. Thanks to Christian Hammers from Debian Security Team for reporting this. (CVE-2004-0388)
- `SHOW GRANTS` with no `FOR` clause or with `FOR CURRENT_USER()` shows the privileges for the current session.

Bugs Fixed

- **Packaging:** In the Mac OS DMG `postinstall` script, `mysql_install_db` was invoked with an invalid argument.
- **Replication:** Replication: a rare race condition in the slave SQL thread that could lead to an incorrect complaint that the relay log is corrupted. (Bug #2011)
- **Replication:** Removed a misleading "check permissions on master.info" from a replication error message, because the cause of the problem could be different from permissions. (Bug #2121)
- **Replication:** A MySQL slave server built using `--with-debug` [98], and replicating itself, crashed. (Bug #3568)

- **Replication:** Multiple-table `DELETE` statements were never replicated by the slave if there were any `--replicate-*-table` options. (Bug #2527)
- **Replication:** If `server-id` was not set using startup options but with `SET GLOBAL`, the replication slave still complained that it was not set. (Bug #3829)
- **Replication:** If a replication slave was unable to create the first relay log, it crashed. (Bug #2145)
- **Replication:** In some replication error messages, a very long query caused the rest of the message to be invisible (truncated), by putting the query last in the message. (Bug #3357)
- **Replication:** Memory could be corrupted by replicating a `LOAD DATA INFILE` from a MySQL 3.23 master. Some less critical issues remain; see Section 14.7, “Replication Features and Issues”. (Bug #3422)
- **Replication:** Replication: in the slave SQL thread, a multiple-table `UPDATE` could produce an incorrect complaint that some record was not found in one table, if the `UPDATE` was preceded by a `INSERT ... SELECT`. (Bug #1701)
- **Replication:** Multiple-table `DELETE` statements were always replicated by the slave if there were some `--replicate-*-ignore-table` options and no `--replicate-*-do-table` options. (Bug #3461)
- **Replication:** Following a nonfatal error during the execution of a statement that later succeeded, the master failed to reset the error code to 0, so the error code was written into the binary log. This caused false `Did not get the same error as on master` errors on the slave. (Bug #2083)
- **Replication:** Corrected the master's binary log position that `InnoDB` reports when it is doing a crash recovery on a slave server. (Bug #3015)
- **Replication:** Changed that when a thread handling `INSERT DELAYED` (also known as a `delayed_insert` thread) is killed, its statements are recorded with an error code of value zero (killing such a thread does not endanger replication, so we thus avoid a superfluous error on the slave). (Bug #3081)
- **Replication:** `CREATE TABLE ... LIKE ...` statements were not always written to the binary log. (Bug #2557)
- **Replication:** Replication: If a client connects to a slave server and issues an administrative statement for a table (for example, `OPTIMIZE TABLE` or `REPAIR TABLE`), this could sometimes stop the slave SQL thread. This does not lead to any corruption, but you must use `START SLAVE` to get replication going again. (Bug #1858)
- **Replication:** `--replicate-wild-*-table` rules now apply to `ALTER DATABASE` when the table pattern is `%`, as is the case for `CREATE DATABASE` and `DROP DATABASE`. (Bug #3000)
- **Replication:** Statements did not raise errors on the slave, if the slave was excluded given the `--replicate-*` options in use at the time. The effect of this problem was: when a statement was killed on the master, the slave stopped. (Bug #2983)
- A rare error condition caused the slave SQL thread spuriously to print the message `Binlog has bad magic number` and stop when it was not necessary to do so. (Bug #3401)
- Queries with subqueries in the `FROM` clause now lock all tables at once. `EXPLAIN` of subqueries in `FROM` output was also not handled correctly. (Bug #2120)
- Prepare statements parameter do not cause error message as fields used in select list but not included in `ORDER BY` list.

- Attempting to bind a negative value bind to unsigned caused an `Unknown error`. (Bug #3223)
- `UNION` statements did not consult `sql_select_limit` [430] value when set. This is now fixed properly, which means that this limit is applied to the top level query, unless `LIMIT` for entire `UNION` is used.
- The `GROUP_CONCAT()` [883] had a number of issues with `ORDER BY` and `DISTINCT`, and with `GROUP BY` in subqueries. (Bug #2695, Bug #3319, Bug #3381)
- Full-text indexing of strings in multi-byte (all besides `utf8`) charsets could sometimes hang. (Bug #2065)
- `SHOW GRANTS` and `EXPLAIN SELECT` did not always perform character set conversion correctly. (Bug #3403)
- `REPAIR TABLE` could corrupt a table containing `FULLTEXT` indexes and many words of different lengths that are considered equal (which is possible in certain collations, such as `latin1_german2_ci` or `utf8_general_ci`). (Bug #3835)
- Processing of `RAND()` [822] in subqueries with static tables was not always handled correctly. (Bug #2645)
- When a `Rotate` event was found by the slave SQL thread in the middle of a transaction, the value of `Relay_Log_Pos` in `SHOW SLAVE STATUS` was incorrectly altered. (Bug #3017)
- `Index_length` in `HEAP` table status for `BTREE` indexes was not calculated correctly. (Bug #2719)
- The `Exec_master_log_pos` column and its disk image in the `relay-log.info` were not handled correctly if the master had version 3.23. (The value was too big by six bytes.) This bug does not exist in MySQL 5.0. (Bug #3400)
- `mysql_stmt_fetch()` and `mysql_stmt_store_result()` could hang if they were called without a prior call to `mysql_stmt_execute()`. Now they give an error instead. (Bug #2248)
- `CHECK TABLE` sometimes produced a spurious error `Found key at page ... that points to record outside datafile` for a table with a `FULLTEXT` index. (Bug #2190)
- `myisamchk` and `CHECK TABLE` that sometimes a spurious error `Found key at page ... that points to record outside datafile` for a table with a `FULLTEXT` index. (Bug #1977)
- Optimization of `ALL` and `SOME` subqueries was not performed well (key field present in subquery). (Bug #3646)
- Starting `mysqld` with binary logging disabled, but with a nonzero value for the `expire_logs_days` [411] system variable caused the server to crash. (Bug #3807)
- Full-text search on multi-byte character sets (such as UTF8) that appeared when a search word was shorter than a matching word from the index (for example, searching for “Uppsala” when the table contains “Uppsa*la”). (Bug #3011)
- On Linux platforms, setting the `character_set_results` [408] variable to `NULL` and then attempting to retrieve it using `SELECT @@character_set_results` caused the server to crash. (Bug #3296)
- The output of `mysqldump --tab` was not correct. (Bug #2705)
- Added support for unsigned integer types to prepared statement API . (Bug #3035)
- An issue with the range optimizer caused a segmentation fault on some very rare queries. (Bug #2698)
- The `INTERVAL()` [785] function did not work correctly when 8 or more comparison arguments were used. (Bug #1561)

- Made clearer the error message that one gets when an update is refused because of the `--read-only` [425] option. (Bug #2757)
- UTF8 charset breaks joins with mixed column/string constant. (Bug #2959)
- Prepared statements are supported for `INSERT`, `REPLACE`, `CREATE`, `DELETE`, `SELECT`, `DO`, `SET` and `SHOW` statements. All other statements are now prohibited by the prepared statement interface. (Bug #3406, Bug #3398, Bug #2811)
- Running `LOAD DATA FROM MASTER` after `RESET SLAVE` caused a segmentation fault. (Bug #2922)
- `EXPLAIN` should now work correctly with `UNION` queries. (Bug #3639)
- `UNION` operations with the `InnoDB` storage engine, when some columns from one table were used in one `SELECT` statement and some were used in another `SELECT` statement, were not handled correctly. (Bug #2552)
- Incorrect error message when wrong table used in multiple-table `DELETE` statement in prepared statements. (Bug #3411)
- `SHOW CREATE TABLE ...` did not properly double quotation marks. (Bug #2593)
- `CREATE ... SELECT` sometimes created a string column with a multi-byte character set (such as `UTF8`) of insufficient length for holding the data.
- The second execution of a prepared statement using `UNION` caused the server to crash. (Bug #3577)
- Removed try to check `NULL` if index built on column where `NULL` is impossible in `IN` subquery optimization. (Bug #2393)
- Using the `GROUP_CONCAT()` [883] function on an expression with `ORDER BY` as well as an external `ORDER BY` in a query caused the server to crash. (Bug #3752)
- Changed that when a `DROP TEMPORARY TABLE` statement is automatically written to the binary log when a session ends, the statement is recorded with an error code of value zero (this ensures that killing a `SELECT` on the master does not result in a superfluous error on the slave). (Bug #3063)
- Short-form IP addresses used as arguments to `INET_ATON()` [878] were not parsed correctly. (Bug #2310)
- Results of aggregate functions used in subqueries with empty result sets were incorrect. (Bug #3505)
- `mysqlbinlog --read-from-remote-server` now print the exact positions of events in lines beginning with `at #` in the log. (Bug #3214)
- Compiling the server using the `--with-pstack` options did not work with binutils 2.13.90. (Bug #1661)
- Changed the column `Seconds_Behind_Master` in `SHOW SLAVE STATUS` to never show a value of -1. (Bug #2826)
- Packaging: Added missing file `mysql_create_system_tables` to the server RPM package. This bug was fixed for the 4.1.1 RPMs by updating the MySQL-server RPM from `MySQL-server-4.1.1-0` to `MySQL-server-4.1.1-1`. The other RPMs were not affected by this change.
- A memory leak could occur with `INSERT ... ON DUPLICATE KEY UPDATE`. (Bug #2438)
- Privileges were not checked correctly for `ALTER TABLE RENAME`. (Bug #3270)
- The MySQL server did not report any error if a statement (submitted through `mysql_real_query()` or `mysql_stmt_prepare()`) was terminated by garbage characters. This can happen if you pass a

wrong `length` parameter to these functions. The result was that the garbage characters were written into the binary log. (Bug #2703)

- `Max_used_connections` [453] was less than the actual maximum number of connections in use simultaneously.
- Aggregate functions could lead to server crashes when used in prepared statements. (Bug #3360)
- `mysqlbinlog --read-from-remote-server` read all binary logs following the one that was requested. It now stops at the end of the requested file, the same as it does when reading a local binary log. There is an option `--to-last-log` to get the old behavior. (Bug #3204)
- Full-text indexing of UTF8 data did not work correctly. (Bug #2033)
- The `--local-load` option of `mysqlbinlog` now requires an argument.
- `CONCAT_WS()` [795] makes the server die in case of illegal mix of collations. (Bug #3087)
- The `mysql` client program crashed when passed a database name that was longer then expected. (Bug #2221)
- Table names in were quoted in `mysqldump` when using values for the server SQL mode where this was not appropriate. (Bug #2591)
- `ANALYZE TABLE` on a `BDB` table inside a transaction caused the server to hang. (Bug #2342)
- `DROP DATABASE` now reports the number of tables deleted.
- Using an impossible `WHERE` with `PROCEDURE ANALYSE()` caused the server to hang. (Bug #2238)
- Parallel repair (`myisamchk -p`, `myisam_repair_threads` [421]) sometimes failed to repair a table. (Bug #1334)
- Added optimization that enables prepared statements using a large number of tables or tables with a large number of columns to be re-executed significantly faster. (Bug #2050)
- A memory leak occurred in the client library when a statement handle was freed on a closed connection (call to `mysql_stmt_close()` after `mysql_close()`). (Bug #3073)
- Invalid results were returned when `CAST()` [859] was applied to `NULL` to obtain a signed or unsigned integer value. (Bug #2219)
- `FLUSH TABLES` sometimes corrupted table resolution for statements which were prepared before the `FLUSH TABLES` but which were being executed repeatedly afterward. (Bug #3307)
- `GRANT` did not handle table-level privileges correctly. (Bug #2178)
- Invoking `mysql_set_server_option()` caused client/server communications to be broken. (Bug #2207)
- A multiple-table `UPDATE` statement resulted in an error when one of the tables was not updated but was used in the nested query contained therein.
- Subqueries in the `FROM` clause were not always parsed correctly. (Bug #2421)
- There was a symlink vulnerability in the `mysqlbug` script. (Bug #3284)
- Table and column privileges were not loaded on startup. (Bug #2546)

- `UNION` operations did not handle `NULL` columns properly, when a column in the first `SELECT` node was `NOT NULL`. (Bug #2508)
- `vio_timeout()` virtual function was not set for all protocols. This lead to crashes on Windows. (Bug #2025)
- You can now call `mysql_stmt_attr_set(..., STMT_ATTR_UPDATE_MAX_LENGTH)` to tell the client library to update `MYSQL_FIELD->max_length` when doing `mysql_stmt_store_result()`. . (Bug #1647)
- A deadlock ocurred when two `START SLAVE` statements were run at the same time. (Bug #2921)
- Comparison of table and database names when using the `--lower_case_table_names` option was not always performed correctly. (Bug #2880)
- A query that uses both `UNION [DISTINCT]` and `UNION ALL` now works correctly. (Bug #1428)
- `mysql_stmt_send_long_data()` misbehaved on the second execution of a prepared statement when long data had zero length. (Bug #1664)
- Table default character set affects `LONGBLOB` columns. (Bug #2821)
- Requiring `UPDATE` [493] privilege for tables which are not updated in multiple-table `UPDATE` statement in prepared statements.
- Segmentation faults could occur when processing malformed prepared statements. (Bug #2795, Bug #2274)
- `mysqldump` did not quote names containing backtick characters (`) correctly. (Bug #2592)
- The results of a query that used `DISTINCT` and `ORDER BY` by a column's real name, while the column had an alias specified in the `SELECT` clause, were not returned in the correct order. (Bug #3681)
- Compile the `MySQL-client` RPM package against `libreadline` instead of `libedit`. (Bug #2289)
- A password was not checked for changes in `GRANT` accounts until `FLUSH PRIVILEGES` was executed. (Bug #3404)
- Subqueries with `OR` and `AND` did not always work correctly. (Bug #2838)
- `ALTER DATABASE` caused the client to hang if the database did not exist. (Bug #2333)
- `ORDER BY` did not always work correctly with `SMALLINT` columns. (Bug #2147)
- When a password was assigned to an account at the global level and then privileges were granted at the database level (without specifying any password), the existing password was replaced temporarily in memory until the next `FLUSH PRIVILEGES` operation or the server was restarted. (Bug #2953)
- `DOUBLE` and `FLOAT` columns could store out of range values. (Bug #2082)
- A prepared statement using parameters and having a subquery in the `FROM` clause could cause the server to crash. (Bug #3020)
- `MATCH ... AGAINST()` on a phrase search operator with a missing closing double quote caused the server to crash. (Bug #2708)
- `mysql_stmt_affected_rows()` now always returns the number of rows affected by a given statement. (Bug #2247)

- When `ALTER TABLE RENAME`, was used to rename a table with the same name in another database, it silently dropped the destination table if it existed. (Bug #2628)
- The syntax `CONVERT(expr,type)` [859] is now supported again.
- `mysqld` could crash when a table was altered and used at the same time. This was a 4.1.2-specific bug. . (Bug #3643)
- Write operations on a column hvaing a `FULLTEXT` index could under some rare circumstances lead to table file corruption. (Bug #2417)
- `mysqlbinlog` failed to print a `USE` statement under those rare circumstances where the binary log contained a `LOAD DATA INFILE` statement. (Bug #3415)

C.1.25 Changes in MySQL 4.1.1 (2003-12-01)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This release includes all fixes in MySQL 4.0.16 and most of the fixes in MySQL 4.0.17.

Functionality Added or Changed

- **Incompatible Change:** Renamed the C API `mysql_prepare_result()` function to `mysql_get_metadata()` because the old name was confusing.
- **Incompatible Change:** Client authentication now is based on 41-byte passwords in the `user` table, not 45-byte passwords as in 4.1.0. Any 45-byte passwords created for 4.1.0 must be reset after running the `mysql_fix_privilege_tables` script.
- **Replication:** Replication over SSL now works.
- **Replication:** `ANALYZE TABLE`, `OPTIMIZE TABLE`, `REPAIR TABLE`, and `FLUSH` statements are now stored in the binary log and thus replicated to slaves. This logging does not occur if the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is given. Exceptions are that `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` are not logged in any case. For a syntax example, see Section 12.4.6.2, “FLUSH Syntax”.
- Added `SQLSTATE` codes for all server errors.
- New `CHECKSUM TABLE` statement for reporting table checksum values.
- Added new `type` values `DAY_MICROSECOND`, `HOUR_MICROSECOND`, `MINUTE_MICROSECOND`, `SECOND_MICROSECOND`, and `MICROSECOND` for `DATE_ADD()` [829], `DATE_SUB()` [833], and `EXTRACT()` [834].
- `TIME` columns with hour values greater than 24 were returned incorrectly to the client.
- Added new syntax for `ADDDATE()` [827] and `SUBDATE()` [839]. The second argument now may be a number representing the number of days to be added to or subtracted from the first date argument.
- Disabled the `PURGE LOGS` statement that was added in version 4.1.0. The statement now should be issued as `PURGE MASTER LOGS` or `PURGE BINARY LOGS`.
- Added the OLAP (On-Line Analytical Processing) function `ROLLUP`, which provides summary rows for each `GROUP BY` level.

- You can revoke all privileges from a user with `REVOKE ALL PRIVILEGES, GRANT FROM user_list`.
- `LIMIT` no longer accepts negative arguments (they used to be treated as very big positive numbers before).
- New `COERCIBILITY()` [871] function to return the collation coercibility of a string.
- Added `DROP USER 'user_name'@'host_name'` statement to drop an account that has no privileges.
- Added new `COMPRESS()` [866], `UNCOMPRESS()` [869], and `UNCOMPRESSED_LENGTH()` [869] functions.
- New global system variable `relay_log_purge` [426] to enable or disable automatic relay log purging.
- `CREATE TABLE tbl_name (...) TYPE=storage_engine` now generates a warning if the named storage engine is not available. The table is still created as a `MyISAM` table, as before.
- Added new `ADDTIME()` [828], `DATE()` [829], `DATEDIFF()` [829], `LAST_DAY()` [836], `MAKEDATE()` [836], `MAKETIME()` [836], `MICROSECOND()` [837], `SUBTIME()` [840], `TIME()` [840], `TIMEDIFF()` [840], `TIMESTAMP()` [840], `UTC_DATE()` [842], `UTC_TIME()` [842], `UTC_TIMESTAMP()` [843], and `WEEKOFYEAR()` [844] functions.
- It is now possible to create a `MERGE` table from `MyISAM` tables in different databases. Formerly, all the `MyISAM` tables had to be in the same database, and the `MERGE` table had to be created in that database as well.
- All queries in which at least one `SELECT` does not use indexes properly now are written to the slow query log when long log format is used.
- `MyISAM` tables now use a better checksum algorithm (if checksum is enabled with `CREATE TABLE ... CHECKSUM = 1`). Old tables will appear to have incorrect checksum, and should be repaired.
- Added `PURGE BINARY LOGS` as an alias for `PURGE MASTER LOGS`.
- Produce warnings even for single-row `INSERT` statements, not just for multiple-row `INSERT` statements. Previously, it was necessary to set `sql_warnings = 1` [430] to generate warnings for single-row statements.
- The `--quote-names` [298] option for `mysqldump` now is enabled by default.
- Table aliases are not case sensitive if `lower_case_table_names` [418] is nonzero.
- Changed that the relay log is flushed to disk by the slave I/O thread every time it reads a relay log event. This reduces the risk of losing some part of the relay log in case of brutal crash.
- Added `secure_auth` [426] global server system variable and `--secure-auth` [392] server option that disallow authentication for accounts that have old (pre-4.1.1) passwords.
- Added new `%f` microseconds format specifier for `DATE_FORMAT()` [832] and `TIME_FORMAT()` [840].
- Phrase search in `MATCH ... AGAINST ( ... IN BOOLEAN MODE)` no longer matches partial words.
- `mysqldump` now includes a statement in the dump output to set `foreign_key_checks` [411] to 0 to avoid problems with tables having to be reloaded in a particular order when the dump is reloaded. The existing `foreign_key_checks` [411] value is saved and restored.
- Added `delimiter` (`\d`) command to the `mysql` command-line client for changing the statement delimiter (terminator). The default delimiter is semicolon.

- Added `preload_buffer_size` [423] system variable.
- The interface to aggregate user-defined functions has changed a bit. You must now declare a `xxx_clear()` function for each aggregate function `XXX()`. `xxx_clear()` is used instead of `xxx_reset()`.
- Added `MATCH ... AGAINST( ... WITH QUERY EXPANSION)` and the `ft_query_expansion_limit` [412] system variable.
- Added `mysql_set_server_option()` C API client function to enable multiple statement handling in the server to be enabled or disabled.
- Added `Slave_IO_State` and `Seconds_Behind_Master` columns to the output of `SHOW SLAVE STATUS`. `Slave_IO_State` indicates the state of the slave I/O thread, and `Seconds_Behind_Master` indicates the number of seconds by which the slave is late compared to the master.
- MySQL source distributions now also include the MySQL Internals Manual `internals.texi`.
- Most subqueries are now much faster than before.
- The `START SLAVE` statement now supports an `UNTIL` clause for specifying that the slave SQL thread should be started but run only until it reaches a given position in the master's binary logs or in the slave's relay logs.
- Added support for syntax `CREATE TABLE table2 (LIKE table1)` that creates an empty table `table2` with a definition that is exactly the same as `table1`, including any indexes.
- `LOAD DATA` now produces warnings that can be fetched with `SHOW WARNINGS`.
- The `--lower-case-table-names=1` [418] server option now also makes aliases case insensitive. (Bug #534)
- Renamed `bdb_version` [407] system variable to `version_bdb` [433].
- Added `mysql_sqlstate()` and `mysql_stmt_sqlstate()` C API client functions that return the `SQLSTATE` error code for the last error.
- Added `IGNORE` option for `DELETE` statement.
- `EXPLAIN` now supports an `EXTENDED` option. When given, `EXPLAIN` generates extra information that may be viewed with the `SHOW WARNINGS` statement.
- Full-text search now supports multi-byte character sets and the Unicode `utf8` character set. (The Unicode `ucs2` character set is not yet supported.)
- Added `SHOW MASTER LOGS` as an alias for `SHOW BINARY LOGS`. (In 4.1.0, `SHOW MASTER LOGS` was renamed to `SHOW BINARY LOGS`. Now you can use either one.)
- Added `--sql-mode=NO_AUTO_VALUE_ON_ZERO` [394] option to suppress the usual behavior of generating the next sequence number when zero is stored in an `AUTO_INCREMENT` column. With this mode enabled, zero is stored as zero; only storing `NULL` generates a sequence number.
- Require `DEFAULT` before table and database default character set. This enables us to use `ALTER TABLE tbl_name ... CHARACTER SET=...` to change the character set for all `CHAR`, `VARCHAR`, and `TEXT` columns in a table.
- Added aggregate function `BIT_XOR()` [882] for bitwise XOR operations.
- Removed unused `ft_max_word_len_for_sort` system variable.

- Removed unused `ft_max_word_len_for_sort` variable from `myisamchk`.
- The `DATABASE()` [872] function now returns `NULL` rather than the empty string if there is no database selected.
- Added `character_set_client` [408], `character_set_connection` [408], `character_set_database` [408], `character_set_results` [408], `character_set_server` [408], `character_set_system` [409], `collation_connection` [409], `collation_database` [409], and `collation_server` [409] system variables to provide information about character sets and collations.
- Added `SHOW BDB LOGS` as an alias for `SHOW LOGS`.
- It is now possible to create multiple key caches, assign table indexes to particular caches, and to preload indexes into caches. See Section 12.4.6.1, "`CACHE INDEX` Syntax". See Section 12.4.6.4, "`LOAD INDEX INTO CACHE` Syntax". Structured system variables are introduced as a means of grouping related key cache parameters. See Section 5.1.4.1, "Structured System Variables".
- The `mysql_next_result()` C API function now returns `-1` if there are no more result sets.
- Added `--secure-auth` option to `mysql` command-line client. If this option is set, the client refuses to send passwords in old (pre-4.1.1) format.
- Renamed `CLIENT_MULTI_QUERIES` connect option flag to `CLIENT_MULTI_STATEMENTS`. To permit a transition period, the old option continues to be recognized for a while.
- `CHAR`, `VARCHAR`, and `TEXT` columns now have lengths measured in characters rather than in bytes. The character size depends on the column's character set. This means, for example, that a `CHAR(n)` column for a multi-byte character set takes more storage than before. Similarly, index values on such columns are measured in characters, not bytes.
- When using `SET sql_mode='mode'` for a complex mode (such as `ANSI` [460]), we now update the `sql_mode` [429] variable to include all the individual options implied by the complex mode.
- The `--old-protocol` [391] option for `mysqld` is no longer supported and has been removed.

Bugs Fixed

- **Security Fix:** A server compiled without SSL support still permitted connections by users who had the `REQUIRE SSL` option specified for their accounts.
- **Security Fix:** Connections from some IP addresses were assigned incorrect database-level privileges. A connection could be assigned the database privileges of the previous successful authentication from one of those IP addresses, even if the IP address user name and database name were different. (Bug #1636)
- **Replication:** The new `PASSWORD()` [868] function in 4.1 is now properly replicated. (Bug #344)
- **Replication:** When an undefined user variable was used in a updating query on the master (such as `INSERT INTO t VALUES(@a)`, where `@a` had never been set by this connection before), the slave could replicate the query incorrectly if a previous transaction on the master used a user variable of the same name. (Bug #1331)
- **Replication:** `CONNECTION_ID()` [872] now is properly replicated. (Bug #177)
- **Replication:** Replication failed between a 3.23 master and a 4.0 slave. The slave lost replicated temporary tables if `FLUSH LOGS` was issued on the master. (Bug #254)

- **Replication:** When a transaction spanned two or more relay logs, and the slave was stopped while executing the part of the transaction that was in the second or later relay log, replication resumed at the beginning of the second or later relay log, which was incorrect. (It should resume at `BEGIN`, in the first relay log.) (Bug #53)
- `LAST_INSERT_ID()` [874] now returns 0 if the last `INSERT` statement didn't insert any rows.
- `HASH`, `BTREE`, `RTREE`, `ERRORS`, and `WARNINGS` no longer are reserved words. (Bug #724)
- A memory overrun could occur due to in subqueries in the `SELECT` list with `WHERE` clause larger than that of the outer query's `WHERE` clause. (Bug #726)
- Using the `?` prepared statement parameter as the argument to certain functions or statement clauses caused a server crash when `mysql_prepare()` was invoked. (Bug #1500)
- Error-handling functions were not called properly when an error resulted from `[CREATE | REPLACE| INSERT] ... SELECT` statements.
- `REPAIR TABLE ... USE_FRM` could cause data loss when used with tables that contained `TIMESTAMP` columns and were created in 4.0.x.
- A `SELECT` that required a temporary table (marked by `Using temporary` in `EXPLAIN` output) and was used as a derived table in `EXPLAIN` statement caused the server to crash. (Bug #251)
- Using `EXPLAIN` on a derived table with a join caused the server to crash.
- The `USER()` [876] function occasionally failed due an error in the size of the string allocated to it.
- `mysql` parser erroneously interpreted a `;` character within a multi-line comment (`/* ... */`) as a statement terminator.
- The types and lengths of result set columns for `UNION` operations are now determined taking into account values for all `SELECT` statements in the `UNION`, and not just the first `SELECT`.
- `ROLLUP` did not work correctly when all tables in the join were `const` [567] tables. (Bug #714)
- The final character was omitted from the output of `USER()` [876]. (Bug #447)
- `DELETE` with `ORDER BY` and `LIMIT` could cause the server to crash.
- Subqueries in `ORDER BY` and `GROUP BY` clauses were not processed correctly. (Bug #442)
- Under certain, rare circumstances table corruption was caused by a `DELETE` from a large table with a “new” (created by MySQL-4.1) full-text index.
- `UNION` with an empty select list and a nonexistent column being used in some of the individual `SELECT` statements could cause the server to crash.
- `SLAVE START` (which is a deprecated syntax, `START SLAVE` should be used instead) could crash the slave. (Bug #2516)
- `UNION` operations that involved temporary tables could cause the server to crash.
- Attempting to create a table containing a spatial (GIS) column using a storage engine that does not support spatial types cause the server to crash.
- When no host name is specified in `SET PASSWORD FOR user`, it now defaults to `%` instead of the current host.
- `MyISAM` tables with `FULLTEXT` indexes created in MySQL 4.0 were unreadable by MySQL 4.1.

- `CREATE FULLTEXT INDEX` was not supported.
- Double the required amount of memory was freed by the server.
- Names of outer columns of subqueries in `INSERT`/`REPLACE` statements were not resolved correctly. (Bug #446)
- Name resolution of columns of reduced subqueries in `UNION` statements was not always performed correctly. (Bug #745)
- Columns of reduced subqueries were not always handled correctly. (Bug #679)
- Following a call to `mysql_prepare()`, placeholders were permitted in all consequent statements, even if they were not prepared. (Bug #1946)
- Privileges could be escalation using database wildcards in `GRANT` statements. (Bug #3924)
- When `ALTER TABLE RENAME`, was used to rename a table with the same name in another database, it silently dropped the destination table if it existed. (Bug #2628)
- A problem with `UNION` kept `NULL` values from being inserted into result set columns where the first `SELECT` of the `UNION` retrieved `NOT NULL` columns. The type and maximum length of the result column are now defined based on all parts of the `UNION`.

C.1.26 Changes in MySQL 4.1.0 (2003-04-03, Alpha)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality Added or Changed

- **Incompatible Change:** `TIMESTAMP` is now returned as a string of type `'YYYY-MM-DD HH:MM:SS'` and different timestamp lengths are not supported.

  This change was necessary for SQL standards compliance. In a future version, a further change will be made (backward compatible with this change), permitting the timestamp length to indicate the desired number of digits of fractions of a second.
- **Replication:** Replication now works with `RAND()` [822] and user variables `@var`.
- The `--opt` [297] option for `mysqldump` now is enabled by default, as are all the options implied by `--opt` [297].
- `SLAVE START` and `SLAVE STOP` are no longer accepted by the query parser; use `START SLAVE` and `STOP SLAVE` instead.
- On Windows, we are now using shared memory to communicate between server and client when they are running on the same machine and you are connecting to `localhost`.
- If one creates a too long `CHAR`/`VARCHAR` it is now automatically changed to `TEXT` or `BLOB`; One get a warning in this case.
- `BTREE` index on `MEMORY` (`HEAP`) tables.
- New `CRC32()` [819] function to compute cyclic redundancy check value.
- Added new `mysql_get_server_version()` C API client function.

- Character sets to be defined per column, table and database.
- `SHOW FULL COLUMNS FROM tbl_name` shows column comments.
- Permit empty index lists to be specified for `USE INDEX`, `IGNORE INDEX`, and `FORCE INDEX`.
- Server side help for all MySQL functions. One can now type `help week` in the `mysql` client and get help for the `WEEK()` function.
- `EXPLAIN SELECT` now can be killed. See Section 12.4.6.3, “`KILL` Syntax”.
- `SELECT ... FROM DUAL` is an alias for `SELECT ...`. (To be compatible with some other database systems).
- New function `IS_USED_LOCK()` [879] for determining the connection identifier of the client that holds a given advisory lock.
- `SERIAL DEFAULT VALUE` added as an alias for `AUTO_INCREMENT`.
- One can add a comment per column in `CREATE TABLE`.
- One can create a table from the existing table using `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table (LIKE table)`. The table can be either normal or temporary.
- `SELECT .. LIMIT 0` did not return the proper row count for `SQL_CALC_FOUND_ROWS`.
- `DROP TEMPORARY TABLE` now drops only temporary tables and doesn't end transactions.
- Added `--compatible` [293] option to `mysqldump` for producing output that is compatible with other database systems or with older MySQL servers.
- Derived tables:

```
SELECT a.col1, b.col2
       FROM (SELECT MAX(col1) AS col1 FROM root_table) a,
       other_table b
       WHERE a.col1=b.col1;
```

- `REPAIR TABLE` of `MyISAM` tables now uses less temporary disk space when sorting char columns.
- `DATE`/`DATETIME` checking is now a bit stricter to support the ability to automatically distinguish between date, datetime, and time with microseconds. For example, dates of type `YYYYMMDD HHMMDD` are no longer supported; you must either have separators between each `DATE`/`TIME` part or not at all.
- `TRUE` and `FALSE` added as alias for 1 and 0, respectively.
- New options `--reconnect` [264] and `--skip-reconnect` [264] for the `mysql` client, to reconnect automatically or not if the connection is lost.
- `START SLAVE` (`STOP SLAVE`) no longer returns an error if the slave is started (stopped); it returns a warning instead.
- Subqueries: `SELECT * from t1 where t1.a=(SELECT t2.b FROM t2)`.
- New more secure client authentication based on 45-byte passwords in the `user` table. (CVE-2000-0981)
- One can specify the different `BLOB`/`TEXT` types with the syntax `BLOB(length)` and `TEXT(length)`. MySQL automatically changes it to one of the internal `BLOB`/`TEXT` types.
- The `mysql` command-line client attempted to interpret quotation marks within comments. (Bug #539)

- In `CREATE TABLE` the attribute `SERIAL` is now an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.
- Added `old-password` command to `mysqladmin` for changing password but storing it using the old password-hashing format.
- `ALTER DATABASE`.
- New `CONVERT(... USING ...)` [859] syntax for converting string values between character sets.
- Multi-line queries: You can now issue several queries at once and then read the results in one go.
- *`expr`* `SOUNDS LIKE` *`expr`* [801] same as `SOUNDEX(`*`expr`*`)=SOUNDEX(`*`expr`*`)` [801].
- One can specify a data type for a column in `CREATE TABLE ... SELECT` by defining the column in the `CREATE TABLE` part.

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

- New operators `integer MOD integer` and `integer DIV integer`. `DIV` is now a reserved word.
- `SHOW [COUNT(*)] WARNINGS` shows warnings from the last command.
- Added support for `UNION` in derived tables.
- `VARCHARACTER` is an alias for `VARCHAR`.
- `CHAR BYTE` is an alias for the `CHAR BINARY` data type.
- In `CREATE TABLE foo (a INT not null primary key)` the `PRIMARY` word is now optional.
- Added new `VARIANCE(`*`expr`*`)` [884] function that returns the variance of *`expr`*
- New `CHARSET()` [870] and `COLLATION()` [871] functions to return the character set and collation of a string.
- Added `record_in_range()` method to `MERGE` tables to be able to choose the correct index when there are many to choose from.
- `REPAIR TABLE` and `OPTIMIZE TABLE` now can be killed. See Section 12.4.6.3, “`KILL` Syntax”.
- Support for GIS (Geometrical data). See Chapter 16, *Spatial Extensions*.
- `libmysqlclient` did not always fetch column default values correctly.
- Aliases are now forced in derived tables, as per standard SQL.
- New faster client/server protocol that supports prepared statements, bound parameters, and bound result columns, binary transfer of data, warnings.
- One can specify many temporary directories to be used in a round-robin fashion with: `--tmpdir=dirname1:dirname2:dirname3`.
- Faster embedded server (new internal communication protocol).
- Permit the `ANSI_QUOTES` [458] SQL mode to be changed on the fly.
- Unicode (UTF8) support.
- Renamed `SHOW MASTER LOGS` statement to `SHOW BINARY LOGS`.

- Added database and real table name (in case of alias) to the `MYSQL_FIELD` structure.
- Permit `DEFAULT(col_name)` [877] in expressions; it produces the column's default value.
- Permit index type to be specified explicitly for some storage engines using `USING type_name` syntax in index definition.

C.2 Changes in Release 4.0.x (Lifecycle Support Ended)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

The following list summarizes the features in MySQL Server 4.0 that are not present in previous versions. For a full list of changes, please refer to the changelog sections for individual 4.0 releases.

- The `InnoDB` storage engine is now included in the standard binaries, adding transactions, row-level locking, and foreign keys. See Section 13.2, “The `InnoDB` Storage Engine”.
- A query cache, offering vastly increased performance for many applications. By caching complete result sets, later identical queries can return instantly. See Section 7.5.3, “The MySQL Query Cache”.
- Improved full-text indexing with boolean mode, truncation, and phrase searching. See Section 11.9, “Full-Text Search Functions”.
- Enhanced `MERGE` tables, now supporting `INSERT` statements and `AUTO_INCREMENT`. See Section 13.3, “The `MERGE` Storage Engine”.
- `UNION` syntax in `SELECT`. See Section 12.2.7.3, “`UNION` Syntax”.
- Multiple-table `DELETE` statements. See Section 12.2.1, “`DELETE` Syntax”.
- `libmysqld`, the embedded server library. See Section 17.5, “libmysqld, the Embedded MySQL Server Library”.
- Additional `GRANT` privilege options for even tighter control and security. See Section 12.4.1.2, “`GRANT` Syntax”.
- Management of user resources in the `GRANT` system, particularly useful for ISPs and other hosting providers. See Section 5.6.4, “Setting Account Resource Limits”.
- Dynamic server variables, allowing configuration changes to be made without having to stop and restart the server. See Section 12.4.4, “`SET` Syntax”.
- Improved replication code and features. See Chapter 14, *Replication*.
- Numerous new functions and options.
- Changes to existing code for enhanced performance and reliability.

For a full list of changes, please refer to the changelog sections for each individual 4.0.x release.

C.2.1 Changes in Release 4.0.31 (Not released)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.0 release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details, please see http://www.mysql.com/products/enterprise.

Bugs fixed:

- **Security Fix**: Using `RENAME TABLE` against a table with explicit `DATA DIRECTORY` and `INDEX DIRECTORY` options can be used to overwrite system table information by replacing the symbolic link points. the file to which the symlink points.

  MySQL will now return an error when the file to which the symlink points already exists. (Bug #321111, CVE-2007-5969)

- Error returns from the `time()` system call were ignored. (Bug #27198)

C.2.2 Changes in Release 4.0.30 (12 February 2007)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.0 release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details, please see http://www.mysql.com/products/enterprise.

Bugs fixed:

- Idle connections were not killed during timeout when using the Native POSIX Thread Library (NPTL) and `mysqld`. In the course of this fix, code to detect and handle the NPTL has been backported from 4.1 to 4.0. (Bug #16995)

C.2.3 Changes in Release 4.0.29 (Not released)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.0 release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details, please see http://www.mysql.com/products/enterprise.

Bugs fixed:

- `InnoDB` exhibited thread thrashing with more than 50 concurrent connections under an update-intensive workload. (Bug #22868)

- `InnoDB` showed substandard performance with multiple queries running concurrently. (Bug #15815)
- User-defined variables could consume excess memory, leading to a crash caused by the exhaustion of resources available to the `MEMORY` storage engine, due to the fact that this engine is used by MySQL for variable storage and intermediate results of `GROUP BY` queries. Where `SET` had been used, such a condition could instead give rise to the misleading error message `You may only use constant expressions with SET`, rather than `Out of memory (Needed NNNNNN bytes)`. (Bug #23443)

C.2.4 Changes in Release 4.0.28 (Not released)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a bugfix release for the MySQL 4.0 release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Enterprise* (a commercial MySQL offering). For more details, please see http://www.mysql.com/products/enterprise.

Functionality added or changed:

- The `mysqldumpslow` script has been moved from client RPM packages to server RPM packages. This corrects a problem where `mysqldumpslow` could not be used with a client-only RPM install, because it depends on `my_print_defaults` which is in the server RPM. (Bug #20216)

Bugs fixed:

- Deleting entries from a large `MyISAM` index could cause index corruption when it needed to shrink. Deletes from an index can happen when a record is deleted, when a key changes and must be moved, and when a key must be un-inserted because of a duplicate key. This can also happen in `REPAIR TABLE` when a duplicate key is found and in `myisamchk` when sorting the records by an index. (Bug #22384)
- Transient errors in replication from master to slave may trigger multiple `Got fatal error 1236: 'binlog truncated in the middle of event'` errors on the slave. (Bug #4053)
- A server or network failure with an open client connection would cause the client to hang even though the server was no longer available. (Bug #9678)
- `mysqlhotcopy` did not copy `RAID` directories with names that contained nondecimal hex digits. (It copied only directories containing the characters `0` through `9` and ignored those containing `a` through `f`.) (Bug #18777)
- Using `SELECT` and a table join while running a concurrent `INSERT` operation would join incorrect rows. (Bug #14400)
- A query with a `WHERE` clause containing *`column`* `= ELT(`*`int_value_1, value_list`*`) OR` *`column`* `= ELT(`*`int_value_2, value_list`*`)` could return unexpected results. (Bug #12728)

C.2.5 Changes in Release 4.0.27 (06 May 2006)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please

consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

This is a security fix release and bugfix release for the MySQL 4.0 release family.

This release includes the patches for recently reported security vulnerabilites in the MySQL client/server protocol. We would like to thank Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting these to us.

Functionality added or changed:

- The MySQL-server RPM now explicitly assigns the mysql system user to the mysql user group during the postinstallation process. This corrects an issue with upgrading the server on some Linux distributions whereby a previously existing mysql user was not changed to the mysql group, resulting in wrong groups for files created following the installation. (Bug #12823)
- Better detection of connection timeout for replication servers on Windows enables elimination of extraneous Lost connection errors in the error log. (Bug #5588)

Bugs fixed:

- **Security fix**: A malicious client, using specially crafted invalid login or COM_TABLE_DUMP packets was able to read uninitialized memory, which potentially, though unlikely in MySQL, could have led to an information disclosure. (CVE-2006-1516, CVE-2006-1517) Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting this bug.
- MySQL-shared-compat-4.0.26-0.i386.rpm incorrectly depend on glibc 2.3 and cannot not be installed on a glibc 2.2 system. For MySQL 4.0, use the older MySQL-shared-compat-4.0.25-0.i386.rpm package. (Bug #16539)
- Running myisampack followed by myisamchk with the --unpack [318] option would corrupt the auto_increment key. (Bug #12633)
- When myisamchk needed to rebuild a table, AUTO_INCREMENT information was lost. (Bug #10405)
- Avoid trying to include <asm/atomic.h> when it doesn't work in C++ code. (Bug #13621)
- BIT_COUNT() [863] could return an incorrect value for right table columns in a LEFT JOIN. (Bug #13044)
- MySQL would not compile on Linux distributions that use the tinfo library. (Bug #18912)
- An UPDATE statement which tried to update a column with a name beginning with an asterisk would cause the server to crash. This was because the server would wrongly expand the * character to the list of all table columns, causing the list of columns to become longer than the list of values. Now the server performs this expansion only if the * character is followed by a space. (Bug #16510)
- An INSERT ... SELECT statement between tables in a MERGE set can return errors when statement involves insert into child table from merge table or vice-versa. (Bug #5390)
- Fixed problems with static variables to allow building on Fedora Core 3. (Bug #6554)
- A LIMIT-related optimization failed to take into account that MyISAM table indexes can be disabled, causing Error 124 when it tried to use such an index. (Bug #14616)
- For a table that had been opened with HANDLER OPEN, issuing OPTIMIZE TABLE, ALTER TABLE, or REPAIR TABLE caused a server crash. (Bug #14397)

- Queries of the form `(SELECT ...) ORDER BY ...` were being treated as a `UNION`. This improperly resulted in only distinct values being returned (because `UNION` by default eliminates duplicate results). Also, references to column aliases in `ORDER BY` clauses following parenthesized `SELECT` statements were not resolved properly. (Bug #7672)
- `SELECT DISTINCT` with a `GROUP BY` clause caused a server crash. (Bug #13855)
- `SHOW CREATE TABLE` did not display any `FOREIGN KEY` clauses if a temporary file could not be created. Now `SHOW CREATE TABLE` displays an error message in an SQL comment if this occurs. (Bug #13002)
- MySQL programs in binary distributions for Solaris 8/9/10 x86 systems would not run on Pentium III machines. (Bug #6772)
- Queries against a `MERGE` table that has a composite index could produce incorrect results. (Bug #9112)
- The counters for the `Key_read_requests` [452], `Key_reads` [452], `Key_write_requests` [452], and `Key_writes` [452] status variables were changed from `unsigned long` to `unsigned longlong` to accommodate larger values before the variables roll over and restart from 0. (Bug #12920)
- A concurrency problem for `CREATE ... SELECT` could cause a server crash. (Bug #12845)
- On HP-UX 11.x (PA-RISC), the `-L` option caused `mysqlimport` to crash. (Bug #12958)
- The server crashed when one thread resized the query cache while another thread was using it. (Bug #12848)

C.2.6 Changes in Release 4.0.26 (08 September 2005)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Added the `mysql_get_client_version()` C API function to the embedded server library. (It was present in the regular client library but inadvertently omitted from the embedded library.) (Bug #10266)

Bugs fixed:

- An optimizer estimate of zero rows for a nonempty `InnoDB` table used in a left or right join could cause incomplete rollback for the table. (Bug #12779)
- Query cache is switched off if a thread (connection) has tables locked. This prevents invalid results where the locking thread inserts values between a second thread connecting and selecting from the table. (Bug #12385)
- For DMG installs on Mac OS X, the preinstallation and postinstallation scripts were being run only for new installations and not for upgrade installations, resulting in an incomplete installation process. (Bug #11380)
- On Windows, applications that used the embedded server made it not possible to remove certain files in the data directory, even after the embedded server had been shut down. This occurred because a file descriptor was being held open. (Bug #12177)
- Creation of the `mysql` group account failed during the RPM installation. (Bug #12348)

- Attempting to repair a table having a fulltext index on a column containing words whose length exceeded 21 characters and where `myisam_repair_threads` [421] was greater than 1 would crash the server. (Bug #11684)
- When two threads compete for the same table, a deadlock could occur if one thread has also a lock on another table through `LOCK TABLES` and the thread is attempting to remove the table in some manner and the other thread want locks on both tables. (Bug #10600)

C.2.7 Changes in Release 4.0.25 (05 July 2005)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Security improvement: Applied a patch to fix a UDF library-loading vulnerability that could result in a buffer overflow and code execution. (CVE-2005-2558)
- Added `--with-big-tables` [98] compilation option to `configure`. (Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable large table support.) See Section 2.9.3, “MySQL Source-Configuration Options”, for details.

Bugs fixed:

- On Mac OS X, `libmysqlclient_r.a` now is built with `--fno-common` to make it possible to link a shared two-level namespace library against `libmysqlclient_r.a`. (Bug #10638)
- An error in the implementation of the `MyISAM` compression algorithm caused `myisampack` to fail with very large sets of data (total size of all the records in a single column needed to be >= 3 GB to trigger this issue). (Bug #8321)
- A problem with the `my_global.h` file caused compilation of MySQL to fail on single-processor Linux systems running 2.6 kernels. (Bug #10364)
- Fixed a portability problem testing for `crypt()` support that caused compilation problems when using OpenSSL/yaSSL on HP-UX and Mac OS X. (Bug #10675, Bug #11150)
- `MyISAM` table corruption could occur with `ANALYZE TABLE` if a write lock was acquired with `LOCK TABLES` and then an `INSERT` or `DELETE` was done prior to analyzing the table. (Bug #10901)
- Fixed a server crash resulting from `CREATE TABLE ... SELECT` that selected from a table being altered by `ALTER TABLE`. (Bug #10224)
- `InnoDB`: In `DROP DATABASE`, check for all referencing tables from other databases before dropping any tables. (Bug #10335)
- Fixed a problem with incorrect constant propagation resulting in incorrect evaluation of `AND` [787] or `OR` [787] queries. (Bug #10095)
- Fixed wrong buffer usage for auto-increment key with blob part that caused `CHECK TABLE` to report that the table was wrong. (Bug #10045)
- No error was raised for `BOOLEAN` full-text searches for storage engines that do not support full-text. (Bug #7709)

- The test in `configure` to see whether `CXX` specified `gcc` failed if `gcc` was specified as a full path name. (Bug #9690)
- In the `mysql_real_escape_string()` C API function, when a multi-byte character is encountered that is illegal in the current character set, escape only the first byte, not each byte. This avoids creating a valid character from an invalid one. (Bug #9864; this is a backport of Bug #8378 from MySQL 4.1.11 to 4.0.25)
- Fixed a deadlock resulting from use of `FLUSH TABLES WITH READ LOCK` while an `INSERT DELAYED` statement is in progress. (Bug #7823)
- Fixed a segmentation fault in `mysqlcheck` that occurred when the last table checked in `--auto-repair` [285] mode returned an error (such as the table being a `MERGE` table). (Bug #9492)
- Fixed faulty display of `TIMESTAMP` columns retrieved as *`col_name`*`+0` while the `new` [390] system variable is set to 1. (Bug #8894)
- Queries containing `CURRENT_USER()` [872] incorrectly were registered in the query cache. (Bug #9796)
- An `UPDATE` that updated only some of the columns in a multiple-column index could result in a loop. (Bug #8942)
- `REPAIR TABLE` did not invalidate query results in the query cache that were generated from the table. (Bug #8480)
- Fixed a bug that caused concurrent inserts to be permitted into the tables in the `SELECT ... UNION ...` part of `INSERT ... SELECT ... UNION ...`. This could result in the incorrect order of queries in the binary log. (Bug #9922)
- Fixed a bug that under certain circumstances could allow a privilege escalation using database wildcards in `GRANT`. (Bug #3924, CVE-2004-0957)
- `<=>` [782] was not properly comparing `NULL` values in the `WHERE` clause of outer joins. (Bug #8711)
- InnoDB: Fixed a bug: MySQL-4.0.23 and 4.0.24 could complain that an InnoDB table created with MySQL-3.23.49 or earlier was in the new compact InnoDB table format of 5.0.3 or later, and InnoDB would refuse to use that table. (The same bug exists in 4.1.8 - 4.1.10.) There is nothing wrong with the table, it is mysqld that is in error. Workaround: wait that 4.0.25 or 4.1.11 is released before doing an upgrade, or dump the table and re-create it with any MySQL version >= 3.23.50 before upgrading to 4.0.23 or 4.0.24.

C.2.8 Changes in Release 4.0.24 (04 March 2005)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Security improvement: The server creates `.frm`, `.MYD`, `.MYI`, `.MRG`, `.ISD`, and `.ISM` table files only if a file with the same name does not already exist. Thanks to Stefano Di Paola `<stefano.dipaola@wisec.it>` for finding and informing us about this issue. (CVE-2005-0711)
- Security improvement: User-defined functions should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. `mysqld` by default no longer loads UDFs unless they have at least one auxiliary symbol defined

in addition to the main symbol. The `--allow-suspicious-udfs` [384] option controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off. `mysqld` also checks UDF file names when it reads them from the `mysql.func` table and rejects those that contain directory path name separator characters. (It already checked names as given in `CREATE FUNCTION` statements.) See Section 18.2.2.1, "UDF Calling Sequences for Simple Functions", Section 18.2.2.2, "UDF Calling Sequences for Aggregate Functions", and Section 18.2.2.6, "User-Defined Function Security Precautions". Thanks to Stefano Di Paola `<stefano.dipaola@wisec.it>` for finding and informing us about this issue. (CVE-2005-0709, CVE-2005-0710)

- `InnoDB`: Added configuration option and settable global variable `innodb_autoextend_increment` [1067] for setting the size in megabytes by which `InnoDB` tablespaces are extended when they become full. The default value is 8, corresponding to the fixed increment of 8MB in previous versions of MySQL.
- InnoDB: Do not acquire an internal `InnoDB` table lock in `LOCK TABLES` if `autocommit = 1` [406]. This helps in porting old `MyISAM` applications to `InnoDB`. `InnoDB` table locks in that case caused deadlocks very easily.

Bugs fixed:

- `AES_DECRYPT(`*`col_name`*`,`*`key`*`)` [865] could fail to return `NULL` for invalid values in *`col_name`*, if *`col_name`* was declared as `NOT NULL`. (Bug #8669)
- `FOUND_ROWS()` [872] returned an incorrect value after a `SELECT SQL_CALC_FOUND_ROWS DISTINCT` statement that selected constants and included `GROUP BY` and `LIMIT` clauses. (Bug #7945)
- Index cardinality was not being updated properly for `TEMPORARY` tables under some circumstances, such as `CREATE TABLE ... SELECT` followed by `ANALYZE TABLE`. (Bug #7519)
- Fixed a server crash caused by `DELETE FROM` *`tbl_name`* `... WHERE ... ORDER BY` *`tbl_name`*`.`*`col_name`* when the `ORDER BY` column was qualified with the table name. (Bug #8392)
- Fixed a bug in `MATCH ... AGAINST` in natural language mode that could cause a server crash if the `FULLTEXT` index was not used in a join (`EXPLAIN` did not show `fulltext` [567] join mode) and the search query matched no rows in the table (Bug #8522).
- Platform and architecture information in version information produced for `--version` option on Windows was always `Win95/Win98 (i32)`. More accurately determine platform as `Win32` or `Win64` for 32-bit or 64-bit Windows, and architecture as `ia32` for x86, `ia64` for Itanium, and `axp` for Alpha. (Bug #4445)
- Fixed an optimization problem that permitted a negative number to be stored in a `DOUBLE UNSIGNED` column when it was assigned a value from a signed `DOUBLE` column. (Bug #7700)
- Fixed a failure of multiple-table updates to replicate properly on slave servers when `--replicate-*-table` options had been specified. (Bug #7011)
- Renamed `set_bit()` and `clear_bit()` functions in source code to avoid a conflict with functions of the same names in Linux kernel header files. (Bug #7971)
- Part of the information being used to cache access-permission lookups was not always reinitialized properly, particularly for connections from localhost on Windows. The result was connection failures that appeared to occur randomly. (Bug #5569)
- Corrected a problem with the `QUOTE()` [800] function returning bad results. (Bug #8248)
- Fixed a problem where `INSERT INTO ...SELECT` failed when the source and target table were the same. (Bug #6034)

- Fixed a problem where RPM installation on Linux as a nonprivileged user would result in incomplete installation. (Bug #7347)
- Change thread stack size used for building Linux RPM distributions to avoid warnings about stack size during server startup. (Bug #6226)
- Fixed a symlink vulnerability in the `mysqlaccess` script. Reported by Javier Fernandez-Sanguino Pena and Debian Security Audit Team. (CVE-2005-0004)
- Fixed support for C API function `mysql_list_fields()`, which was accidentally broken in 4.0.22 (Bug #6761)
- Make `query_cache_wlock_invalidate` [425] system variable visible in `SHOW VARIABLES` output. (Bug #7594)
- Fixed a bug which caused `FROM_UNIXTIME()` [834] function to return `NULL` for zero argument instead of the Epoch. (Bug #7515)
- Now in datetime values two digit year is interpreted as year in 20th or 21st century even with zero month and day. (Bug #7297)
- Fixed a bug in `QUOTE` function when used in conjunction with some other string functions. This lead to severe buffer overflow and server crashing. (Bug #7495)
- InnoDB: Work around a problem in AIX 5.1 patched with ML7 security patch: InnoDB would refuse to open its `ibdata` files, complaining about an operating system error 0.
- InnoDB: Fixed a memory corruption bug if one created a table with a primary key that contained at least two column prefixes. An example: `CREATE TABLE t(a char(100), b tinyblob, PRIMARY KEY(a(5), b(10)))`.
- InnoDB: Use native `tmpfile()` function on Netware. All InnoDB temporary files are created under `sys:\tmp`. Previously, InnoDB temporary files were never deleted on Netware.
- `InnoDB`: Honor the `--tmpdir` [394] startup option when creating temporary files. Previously, `InnoDB` temporary files were always created in the temporary directory of the operating system. On Netware, `InnoDB` will continue to ignore `--tmpdir` [394]. (Bug #5822)
- InnoDB: Fix a theoretical hang over the adaptive hash latch in InnoDB if one runs `INSERT ... SELECT ...` (binlog not enabled), or a multiple-table `UPDATE` or `DELETE`, and only the read tables are InnoDB type, the rest are `MyISAM`; this also fixes bug #7879 for InnoDB type tables. (Bug #7879)
- InnoDB: Fixed a bug: 32-bit `mysqld` binaries built on HP-UX-11 did not work with `InnoDB` files greater than 2 GB in size. (Bug #6189)
- InnoDB: Fixed a bug: InnoDB failed to drop a table in the background drop queue if the table was referenced by a foreign key constraint.
- InnoDB: Fixed a bug: if we dropped a table where an `INSERT` was waiting for a lock to check a `FOREIGN KEY` constraint, then an assertion would fail in `lock_reset_all_on_table()`, since that operation assumes no waiting locks on the table or its records.
- Fixed that, when encountering a “disk full” or “quota exceeded” write error, `MyISAM` sometimes didn't sleep and retry the write, thus resulting in a corrupted table. (Bug #7714)
- Fixed that a slave could crash after replicating many `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `REPAIR TABLE` statements from the master. (Bug #6461, Bug #7658)

- Fixed a bug where MySQL was allowing concurrent updates (inserts, deletes) to a table if binary logging is enabled. Changed to ensure that all updates are executed in a serialized fashion, because they are executed serialized when binlog is replayed. (Bug #7879)
- Fixed a bug in replication that caused the master to stamp generated statements (such as `SET` statements) with an `error_code` intended only for another statement. This could happen, for example, when a statements generates a duplicate key error on the master but must be replicated. (Bug #8412)
- Documented problem with using `mysqldump` in 4.0.x to dump `TIMESTAMP(2)` and `TIMESTAMP(4)` data types. (Bug #6530)

C.2.9 Changes in Release 4.0.23 (18 December 2004)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

Due to a `libtool`-related bug in the source distribution, the creation of shared `libmysqlclient` libraries was not possible (the resulting files were missing the `.so` file name extension). The file `ltmain.sh` was updated to fix this problem and the resulting source distribution was released as `mysql-4.0.23a.tar.gz`. This modification did not affect the binary packages. (Bug #7401)

Functionality added or changed:

- Added `--hex-blob` [295] option to `mysqldump` for dumping binary string columns using hexadecimal notation.
- Added `mysql_hex_string()` C API function that hex-encodes a string.
- InnoDB: Do not periodically write `SHOW INNODB STATUS` information to a temporary file unless the configuration option `innodb-status-file = 1` [1066] is set.
- InnoDB: Made the foreign key parser better aware of quotation marks. (Bug #6340)
- `mysqlbinlog` now prints an informative commented line (thread id, timestamp, server id, and so forth) before each `LOAD DATA INFILE`, like it does for other queries; unless `--short-form` [341] is used.

Bugs fixed:

- A multiple-table `DELETE` could cause MySQL to crash when using `InnoDB` tables. (Bug #5837, Bug #6378)
- Corrected accounts in the `mysql.user` table in Windows distributions that had been created with a `Host` value of `build` rather than `%`. (Bug #6000)
- Prevent adding `CREATE TABLE .. SELECT` query to the binary log when the insertion of new records partially failed. (Bug #6682)
- Fixed bug which caused `FROM_UNIXTIME()` [834] function to return wrong result if the argument was too big. (Bug #6439)
- Fixed bug which caused MySQL server to store wrong values in `TIMESTAMP` columns and give wrong results for `UNIX_TIMESTAMP()` [841] function if it was run in time zone with leap seconds. (Bug #6387)

- InnoDB: Fixed a bug in `LOAD DATA INFILE...REPLACE` printing duplicate key error when executing the same load query several times. (Bug #5835)
- InnoDB: Refuse to open new-style tables created with MySQL 5.0.3 or later. (Bug #7089)
- InnoDB: Do not call `rewind()` when displaying `SHOW INNODB STATUS` information on `stderr`.
- InnoDB: If one used `INSERT IGNORE` to insert several rows at a time, and the first inserts were ignored because of a duplicate key collision, then InnoDB in a replication slave assigned `AUTO_INCREMENT` values 1 bigger than in the master. This broke the MySQL replication. (Bug #6287)
- InnoDB: Fix two hangs: `FOREIGN KEY` constraints treated table and database names as case-insensitive. `RENAME TABLE t TO T` would hang in an endless loop if `t` had a foreign key constraint defined on it. Fix also a hang over the dictionary mutex that would occur if one tried in `ALTER TABLE` or `RENAME TABLE` to create a foreign key constraint name that collided with another existing name. (Bug #3478)
- InnoDB: Treat character `0xA0` as space in InnoDB's `FOREIGN KEY` parser if MySQL treats it as space in the default charset. EMS MySQL Manager inserts character `0xA0` after the table name in an `ALTER`, which confused InnoDB's parser.
- Fixed a bug which caused a crash when only the slave I/O thread was stopped and restarted. (Bug #6148)
- If a connection had an open transaction but had done no updates to transactional tables (for example, if had just done a `SELECT FOR UPDATE` then executed a nontransactional update, that update automatically committed the transaction (thus releasing InnoDB's row-level locks etc). (Bug #5714)
- If a connection was interrupted by a network error and did a rollback, the network error code got stored into the `BEGIN` and `ROLLBACK` binary log events; that caused superfluous slave stops. (Bug #6522)
- A sequence of `BEGIN` (or `SET autocommit = 0`), `FLUSH TABLES WITH READ LOCK`, transactional update, `COMMIT`, `FLUSH TABLES WITH READ LOCK` could hang the connection forever and possibly the MySQL server itself. This happened for example when running the `innobackup` script several times. (Bug #6732)

C.2.10 Changes in Release 4.0.22 (27 October 2004)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- The `--with-openssl` option for `configure` now accepts a path prefix as an argument. `--with-openssl-includes` and `--with-openssl-libs` are still supported, but are needed only to override the default values. (Bug #5494)
- Added new `--without-man` option to `configure` to suppress building/installing the manual pages. (Bug #5379)
- InnoDB: New `mysqld` option session variable `innodb_table_locks` [1072] (on by default). In applications using `autocommit = 1` [406] and MySQL's `LOCK TABLES` statement, InnoDB's internal table locks that were added in 4.0.20 can cause deadlocks. You can set `innodb_table_locks = 0` in `my.cnf` to remove that problem. See Section 13.2.15, "Restrictions on InnoDB Tables". (Bug #3299, Bug #5998)

- InnoDB: Added the startup option and settable global variable `innodb_max_purge_lag` [1071] for delaying `INSERT`, `UPDATE` and `DELETE` operations when the purge operations are lagging. The default value of this parameter is zero, meaning that there are no delays. See Section 13.2.10, “`InnoDB` Multi-Versioning”.

- InnoDB: Change error code to `HA_ERR_ROW_IS_REFERENCED` if we cannot `DROP` a parent table because it is referenced by a `FOREIGN KEY` constraint.

Bugs fixed:

- Fixed bug in server which caused connection stall when one of deprecated `libmysqlclient` functions `mysql_create_db()` and `mysql_rm_db()` were called and were going to return error. (Bug #6081)

- Fixed returning wrong query result from query cache if a temporary table was hiding a real table after putting results to query cache. (Bug #6084)

- Fixed `ENABLE KEYS`, which failed if `tmpdir` [432] ran out of space. Now, a full repair is done in this case. (Bug #5625)

- Fixed an improper error message when trying to drop a table which is referenced by a `FOREIGN KEY` constraint. (Bug #5784)

- Fixed a bug that permitted `FLUSH TABLES` to close `HANDLER` tables. `HANDLER` tables are now reopened after a `FLUSH TABLES` the next time they are used. However, they lose their file position if this happens. (Bug #4286)

- Fixed a bug that permitted `HANDLER` tables with the same alias to be opened multiple times. `HANDLER` aliases must now be unique, even though it is syntactically correct in versions below 4.1 to qualify them with their base table's database name (for example, `test_db.handler_tbl` now conflicts with `another_db.handler_tbl`). (Bug #4335)

- Fixed crash when using MySQL 4.0 with privilege tables from MySQL 5.0.

- `mysqlimport` now reads input files locally from the client host only if the `--local` [304] option is given. Previously, it assumed incorrectly in some cases that files were local even without `--local` [304]. (Bug #5829)

- InnoDB: Make the check for excessive semaphore waits to tolerate glitches in the system clock (do not crash the server if the system time is adjusted while InnoDB is under load.). (Bug #5898)

- InnoDB: Fixed a bug in the InnoDB `FOREIGN KEY` parser that prevented `ALTER TABLE` of tables containing “`#`” in their names. (Bug #5856)

- InnoDB: Fixed problem introduced in 4.0.21 where a connection starting a transaction, doing updates, then `FLUSH TABLES WITH READ LOCK`, then `COMMIT`, would cause replication slaves to stop (complaining about error 1223). Bug surfaced when using the InnoDB `innobackup` script. (Bug #5949)

- InnoDB: If one updated a column so that its size changed, or updated it to an externally stored (`TEXT` or `BLOB`) value, then ANOTHER externally stored column would show up as 512 bytes of good data + 20 bytes of garbage in a consistent read that fetched the old version of the row. (Bug #5960)

- InnoDB: Release the dictionary latch during a long cascaded `FOREIGN KEY` operation, so that we do not starve other users doing `CREATE TABLE` or other DDL operations. This caused a notorious 'Long semaphore wait' message to be printed to the `.err` log. (Bug #5961)

- InnoDB: Let InnoDB remember row locking type (X or S) inside `LOCK TABLES`, also over plain consistent read `SELECT`s.

- InnoDB: Fixed a bug introduced in 4.0.21. An assertion failed if one used `mysqldump` with the option `-l` or `--opt` [297], or if one used `LOCK TABLES ... LOCAL`. (Workaround in 4.0.21: use `--quick` [298] and `--single-transaction` [298]. (Bug #5538)
- InnoDB: Having a column prefix index in the primary key, and the same column fully in a secondary key could cause an assertion failure in row_build_row_ref(). (Bug #5180)
- Fixed a bug which resulted in an erroneously calculated number of examined rows in `UNION`s. This value is printed in the slow query log. (Bug #5879)
- Fixed bug with crash of server on some values of `read_rnd_buffer_size` [426] (Bug #5492)
- Fixed bug which caused truncation of values read from or into `TIMESTAMP` fields if `--new` mode was enabled. (Bug #4131)
- `mysqladmin` now returns a status of 0 even when the server denies access; such an error means the server is running. (Bug #3120)
- Fixed that if the slave SQL thread found a syntax error in a query (which should be rare, as the master parsed it successfully), it stops. (Bug #5711)
- Fixed that if a write to a `MyISAM` table fails because of a full disk or an exceeded disk quota, it prints a message to the error log every 10 minutes, and waits until disk becomes free. (Bug #3248)
- Fixed problem with symlinked databases on Windows being shown with `SHOW DATABASES` even if the database name doesn't match the given wildcard (Bug #5539)

C.2.11 Changes in Release 4.0.21 (06 September 2004)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Print *`version_comment`* (from `./configure --comment` during compilation) when starting the server. Example: `Version: '4.0.21-debug' socket: '/tmp/mysql.sock' port: 0 Official MySQL Binary`
- Made the MySQL server not react to signals `SIGHUP` and `SIGQUIT` on Mac OS X 10.3. This is needed because under this OS, the MySQL server receives lots of these signals (reported as Bug #2030).
- On Windows, the `mysqld-nt` and `mysqld-max-nt` servers now write error messages to the Windows event log in addition to the MySQL error log.

Bugs fixed:

- Fixed an old bug in concurrent accesses to `MERGE` tables (even one `MERGE` table and `MyISAM` tables), that could've resulted in a crash or hang of the server. (Bug #2408, CVE-2004-0837)
- Fixed a bug that caused incorrect results from `GROUP BY` queries with expression in `HAVING` clause that refers to a columns such as `BLOB`, `TEXT`, or `TINYBLOB`. (Bug #4358)
- Fixed a bug when memory was not released when `HEAP` table is dropped. It could only happen on Windows when a symlink file (.sym) is used and if that symlink file contained double backslashes (\\). (Bug #4973)

- Fixed a bug which prevented `TIMESTAMP(19)` fields from being created. (Bug #4491)
- Fixed a bug that caused wrong results in queries that were using index to search for `NULL` values in `BLOB` (`TINYBLOB`, `TEXT`, `TINYTEXT`, etc) columns of `MyISAM` tables. (Bug #4816)
- Fixed a bug in the function `ROUND()` [823] reporting incorrect metadata (number of digits after the decimal point). It can be seen, for example, in `CREATE TABLE t1 SELECT ROUND(1, 34)`. (Bug #4393)
- Fixed precision loss bug in some mathematical functions such as `SQRT()` [824] and `LOG()` [820]. (Bug #4356)
- Fixed a long-standing problem with `LOAD DATA` with the `LOCAL` option. The problem occurs when an error happens during the `LOAD DATA` operation. Previously, the connection was broken. Now the error message is returned and connection stays open.
- Optimizer now treats `col IN (val)` the same way it does for `col = val`.
- Fixed a problem with `net_buffer_length` [422] when building the `DBD::mysql` Perl module. (Bug #4206)
- `lower_case_table_names = 2` (keep case for table names) was not honored with `ALTER TABLE` and `CREATE/DROP INDEX`. (Bug #3109)
- Fixed a crash on declaration of `DECIMAL(0,...)` column. (Bug #4046)
- Fixed a bug in `IF()` [790] function incorrectly determining the result type if aggregate functions were involved. (Bug #3987)
- Fixed bug in privilege checking where, under some conditions, one was able to grant privileges on the database, he has no privileges on. (Bug #3933)
- Fixed crash in `MATCH ... AGAINST()` on a phrase search operator with a missing closing double quote. (Bug #3870, CVE-2004-0956)
- Values greater than 4294967295 of system variables were truncated on 64-bit platforms. (Bug #3754)
- If `server-id` was not set using startup options but with `SET GLOBAL`, the replication slave still complained that it was not set. (Bug #3829)
- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017, CVE-2004-0836)
- During the installation process of the server RPM on Linux, `mysqld` was run as the `root` system user, and if you had `--log-bin=`*`somewhere_out_of_var_lib_mysql`* [1181] it created binary log files owned by `root` in this directory, which remained owned by `root` after the installation. This is now fixed by starting `mysqld` as the `mysql` system user instead. (Bug #4038)
- Made `DROP DATABASE` honor the value of `lower_case_table_names` [418]. (Bug #4066)
- The slave SQL thread refused to replicate `INSERT ... SELECT` if it examined more than 4 billion rows. (Bug #3871)
- Fixed incorrect destruction of expression which led to crash of server on complex `AND` [787]/`OR` [787] expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969, Bug #4494)
- Fixed that `mysqlbinlog --position --read-from-remote-server` had wrong `# at` lines. (Bug #4506)

- If `CREATE TEMPORARY TABLE t SELECT` failed while loading the data, the temporary table was not dropped. (Bug #4551)
- Fixed that when a multiple-table `DROP TABLE` failed to drop a table on the master server, the error code was not written to the binary log. (Bug #4553)
- When the slave SQL thread was replicating a `LOAD DATA INFILE` statement, it didn't show the statement in the output of `SHOW PROCESSLIST`. (Bug #4326)
- Fixed that `CREATE TABLE ... TYPE=HEAP ... AS SELECT...` caused replication slave to stop. (Bug #4971)
- Fixed that `disable-local-infile` option had no effect if client read it from a configuration file using `mysql_options(...,MYSQL_READ_DEFAULT,...)`. (Bug #5073)
- Fixed that `mysql-test-run` failed on the `rpl_trunc_binlog` test if running test from the installed (the target of 'make install') directory. (Bug #5050)
- Fixed an unlikely deadlock which could happen when using `KILL`. (Bug #4810)
- Fixed a crash when one connection got `KILL`ed while it was doing `START SLAVE`. (Bug #4827)
- Made `FLUSH TABLES WITH READ LOCK` block `COMMIT` if server is running with binary logging; this ensures that the binary log position is trustable when doing a full backup of tables and the binary log. (Bug #4953)
- Fixed that the counter of an `auto_increment` column was not reset by `TRUNCATE TABLE` is the table was a temporary one. (Bug #5033)
- Made database names to compare case-insensitively in fully qualified column names (`database.table.column`) when `lower_case_table_names = 1`. (Bug #4792)
- Fixed that `SET CHARACTER SET` was not replicated correctly. MySQL 4.1 does not have that bug. (Bug #4500)
- Fixed a symlink vulnerability in the `mysqlhotcopy` script. (CVE-2004-0457)

C.2.12 Changes in Release 4.0.20 (17 May 2004)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

The windows packages had to be repackaged and re-released several times to resolve packaging issues (such as missing files). This did not affect the binaries included (they have not been recompiled), therefore the installation packages are of version 4.0.20d, while the binaries included still identify themselves as version 4.0.20b.

Functionality added or changed:

- From the Windows distribution, predefined accounts without passwords for remote users ("root@%", "@%") were removed (other distributions never had them).

- Phrase search in `MATCH ... AGAINST ( ... IN BOOLEAN MODE)` no longer matches partial words.

Bugs fixed:

- A crashing bug (race condition) was fixed in InnoDB diagnostic logging. It was introduced in 4.0.19. (Bug #3596)
- Fixed a bug in division `/` reporting incorrect metadata (number of digits after the decimal point). It can be seen, for example, in `CREATE TABLE t1 SELECT "0.01"/"3"`. (Bug #3612)
- Fixed a problem with nonworking `DROP DATABASE` on some configurations (in particular, Linux 2.6.5 with ext3 are known to expose this bug). (Bug #3594)
- Fixed that in some replication error messages, a very long query caused the rest of the message to be invisible (truncated), by putting the query last in the message. (Bug #3357)

C.2.13 Changes in Release 4.0.19 (04 May 2004)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Note

The MySQL 4.0.19 binaries were uploaded to the download mirrors on May, 10th. However, a potential crashing bug was found just before the 4.0.19 release was publicly announced and published from the 4.0 download pages at http://dev.mysql.com/.

A fix for the bug was pushed into the MySQL source tree shortly after it could be reproduced and is included in MySQL 4.0.20. Users upgrading from MySQL 4.0.18 should upgrade directly to MySQL 4.0.20 or later.

See (Bug #3596) for details (it was reported against MySQL-4.1, but was confirmed to affect 4.0.19 as well).

Functionality added or changed:

- If length of a timestamp field is defined as 19, the timestamp is displayed as `"YYYY-MM-DD HH:MM:SS`. This is done to make it easier to use tables created in MySQL 4.1 to be used in MySQL 4.0.
- If you use `RAID_CHUNKS` with a value > 255 it is set to 255. This was made to ensure that all raid directories are always 2 hex bytes. (Bug #3182)
- Changed that the optimizer now considers the index specified in `FORCE INDEX` clause as a candidate to resolve `ORDER BY` as well.
- The `--log-warnings` [389] server option now is enabled by default. Disable with `--log-warnings=0` [389].
- Until now, in `SELECT ... UNION SELECT ... ORDER BY ...`, it was possible to qualify a column name in the `ORDER BY` clause with a table name. This is no longer possible. Column names in `ORDER BY` should refer to names established in the first `SELECT` of the `UNION`. (Bug #3064)
- Added `max_insert_delayed_threads` [419] system variable as a synonym for `max_delayed_threads` [419].

- Added `query_cache_wlock_invalidate` [425] system variable. It enables emulation of `MyISAM` table write-locking behavior, even for queries in the query cache. (Bug #2693)
- The keyword `MASTER_SERVER_ID` is not reserved anymore.
- The following is relevant mainly for Mac OS X users who use a case-insensitive file system. This is not relevant for Windows users as `InnoDB` in this case always stores file names in lower case:

  You can now force `lower_case_table_names` [418] to 0 from the command line or a configuration file. This is useful with case-insensitive file systems when you have previously not used `lower_case_table_names = 1` [418] or `lower_case_table_names = 2` [418] and you have created `InnoDB` tables. With `lower_case_table_names = 0` [418], `InnoDB` tables were stored in mixed case while setting `lower_case_table_names` [418] to a nonzero value now forces it to lower case (to make the table names case insensitive).

  Because it is possible to crash `MyISAM` tables by referring to them with different case on a case-insensitive file system, use `lower_case_table_names` [418] or `lower_case_table_names = 2` [418] on such file systems.

  The easiest way to convert to use `lower_case_table_names = 2` [418] is to dump all your `InnoDB` tables with `mysqldump`, drop them and then restore them.
- Changed that the relay log is flushed to disk by the slave I/O thread every time it reads a relay log event. This reduces the risk of losing some part of the relay log in case of brutal crash.
- When a session having open temporary tables terminates, the statement automatically written to the binary log is now `DROP TEMPORARY TABLE IF EXISTS` instead of `DROP TEMPORARY TABLE`, for more robustness.
- Added option `--replicate-same-server-id` [1177].

Bugs fixed:

- Added missing full-text variable `ft_stopword_file` [412] to `myisamchk`.
- Do not allow stray `','` at the end of field specifications. (Bug #3481)
- `INTERVAL` now can handle big values for seconds, minutes and hours. (Bug #3498)
- Blank host name did not work as documented for table and column privileges. Now it works the same way as `'%'`. (Bug #3473)
- Fixed a harmless buffer overflow in `replace` utility. (Bug #3541)
- Fixed `SOUNDEX()` [801] to ignore nonalphabetic characters also in the beginning of the string. (Bug #3556)
- Fixed a bug in `MATCH ... AGAINST()` searches when another thread was doing concurrent inserts into the `MyISAM` table in question. The first --- full-text search --- query could return incorrect results in this case (for example, “phantom” rows or not all matching rows, even an empty result set). The easiest way to check whether you are affected is to start `mysqld` with `--skip-concurrent-insert` [392] switch and see whether it helps.
- Fixed bug when doing `DROP DATABASE` on a directory containing non- MySQL files. Now a proper error message is returned.
- Fixed bug in `ANALYZE TABLE` on a `BDB` table inside a transaction that hangs server thread. (Bug #2342)

- Fixed a symlink vulnerability in the `mysqlbug` script. (Bug #3284, CVE-2004-0381)
- Fixed core dump bug in `SELECT DISTINCT` where all selected parts where constants and there were hidden columns in the created temporary table. (Bug #3203)
- Fixed core dump bug in `COUNT(DISTINCT)` [882] when there was a lot of values and one had a big value for `max_heap_table_size` [419].
- Fixed problem with multiple-table-update and BDB tables. (Bug: #3098)
- Fixed memory leak when dropping database with `RAID` tables. (Bug #2882)
- Fixed core dump crash in replication during relay-log switch when the relay log went over `max_relay_log_size` [420] and the slave thread did a `flush_io_cache()` at the same time.
- Fixed hangup bug when issuing multiple `SLAVE START` from different threads at the same time. (Bug #2921)
- Fixed bug when using `DROP DATABASE` with `lower_case_table_names = 2` [418].
- Fixed wrong result in `UNION` when using `lower_case_table_names = 2` [418]. (Bug #2858)
- One can now kill threads that is 'stuck' in the join optimizer (can happen when there is MANY tables in the join in which case the optimizer can take really long time). (Bug #2825)
- Rollback `DELETE` and `UPDATE` statements if thread is killed. (Bug #2422)
- Ensure that all rows in an `INSERT DELAYED` statement is written at once if binary logging is enabled. (Bug #2491).
- Fixed bug in query cache statistic, more accurate formula linked statistic variables mentioned in the manual.
- Fixed a bug in parallel repair (`myisamchk -p`, `myisam_repair_threads` [421]) - sometimes repair process failed to repair a table. (Bug #1334)
- Fixed bugs with names of tables, databases, and columns that end to space (Bug #2985)
- Fixed a bug in multiple-table `UPDATE` statements involving at least one constant table. Bug was exhibited in allowing non matching row to be updated. (Bug #2996).
- Fixed all bugs in scripts for creating/upgrading system database (Bug #2874) Added tests which guarantee against such bugs in the future.
- Fixed bug in `mysql` command-line client in interpreting quotation marks within comments. (Bug #539)
- `--set-character-set` [318] and `--character-sets-dir` [317] options in `myisamchk` now work.
- Fixed a bug in `mysqlbinlog` that caused one pointer to be free'd twice in some cases.
- Fixed a bug in boolean full-text search, that sometimes could lead to false matches in queries with several levels of subexpressions using `+` operator (for example, `MATCH ... AGAINST('+(+(word1 word2)) +word3*' IN BOOLEAN MODE)`.
- Fixed Windows-specific portability bugs in `myisam_ftdump`.
- Fixed a bug in multiple-table `DELETE` that was caused by foreign key constraints. If the order of the tables established by MySQL optimizer did not match parent-child order, no rows were deleted and no error message was provided. (Bug #2799)

- Fixed a few years old bug in the range optimizer that caused a segmentation fault on some very rare queries. (Bug #2698)
- Replication: If a client connects to a slave server and issues an administrative statement for a table (for example, `OPTIMIZE TABLE` or `REPAIR TABLE`), this could sometimes stop the slave SQL thread. This does not lead to any corruption, but you must use `START SLAVE` to get replication going again. (Bug #1858) The bug was accidentally not fixed in 4.0.17 as it was unfortunately earlier said.
- Fixed that when a `Rotate` event is found by the slave SQL thread in the middle of a transaction, the value of `Relay_Log_Pos` in `SHOW SLAVE STATUS` remains correct. (Bug #3017)
- Corrected the master's binary log position that `InnoDB` reports when it is doing a crash recovery on a slave server. (Bug #3015)
- Changed that when a `DROP TEMPORARY TABLE` statement is automatically written to the binary log when a session ends, the statement is recorded with an error code of value zero (this ensures that killing a `SELECT` on the master does not result in a superfluous error on the slave). (Bug #3063)
- Changed that when a thread handling `INSERT DELAYED` (also known as a `delayed_insert` thread) is killed, its statements are recorded with an error code of value zero (killing such a thread does not endanger replication, so we thus avoid a superfluous error on the slave). (Bug #3081)
- Fixed deadlock when two `START SLAVE` statements were run at the same time. (Bug #2921)
- Fixed that a statement never triggers a superfluous error on the slave, if it must be excluded given the `--replicate-*` options. The bug was that if the statement had been killed on the master, the slave would stop. (Bug #2983)
- The `--local-load` [340] option of `mysqlbinlog` now requires an argument.
- Fixed a segmentation fault when running `LOAD DATA FROM MASTER` after `RESET SLAVE`. (Bug #2922)
- Fixed a rare error condition that caused the slave SQL thread spuriously to print the message `Binlog has bad magic number` and stop when it was not necessary to do so. (Bug #3401)
- Fixed bug in privilege checking of `ALTER TABLE RENAME`. (Bug #3270, CVE-2004-0835)
- Fixed the column `Exec_master_log_pos` (and its disk image in the `relay-log.info` file) to be correct if the master had version 3.23 (it was too big by 6 bytes). This bug does not exist in the 5.0 version. (Bug #3400)
- Fixed that `mysqlbinlog` does not forget to print a `USE` statement under rare circumstances where the binary log contained a `LOAD DATA INFILE` statement. (Bug #3415)
- Fixed a memory corruption when replicating a `LOAD DATA INFILE` when the master had version 3.23. Some smaller problems remain in this setup, See Section 14.7, “Replication Features and Issues”. (Bug #3422)
- Multiple-table `DELETE` statements were always replicated by the slave if there were some `--replicate-*-ignore-table` options and no `--replicate-*-do-table` options. (Bug #3461)
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` [98] and replicating itself. (Bug #3568)

C.2.14 Changes in Release 4.0.18 (12 February 2004)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please

consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Fixed processing of `LOAD DATA` by `mysqlbinlog` in remote mode. (Bug #1378)
- The `ft_dump` utility program was renamed to `myisam_ftdump`, and is included in binary distributions.
- `ENGINE` is now a synonym for the `TYPE` option for `CREATE TABLE` and `ALTER TABLE`.
- `lower_case_table_names` [418] system variable now can take a value of `2`, to store table names in mixed case on case-insensitive file systems. It is forced to 2 if the database directory is located on a case-insensitive file system.
- For replication of `MEMORY` (`HEAP`) tables: Made the master automatically write a `DELETE FROM` statement to its binary log when a `MEMORY` table is opened for the first time since master's startup. This is for the case where the slave has replicated a nonempty `MEMORY` table, then the master is shut down and restarted: the table is now empty on master; the `DELETE FROM` empties it on slave too. Note that even with this fix, between the master's restart and the first use of the table on master, the slave still has out-of-date data in the table. But if you use the `init-file` option to populate the `MEMORY` table on the master at startup, it ensures that the failing time interval is zero. (Bug #2477)
- Optimizer is now better tuned for the case where the first used key part (of many) is a constant. (Bug #1679)
- Removed old nonworking `--old-rpl-compat` server option, which was a holdover from the very first 4.0.x versions. (Bug #2428)
- Added `sync_frm` [430] system variable. It is enabled by default, to instruct MySQL to sync to disk each time an `.frm` file is created. Disable it to suppress these sync operations.

Bugs fixed:

- `mysqlhotcopy` now works on NetWare.
- `DROP DATABASE` could not drop databases with RAID tables that had more than nine `RAID_CHUNKS`. (Bug #2627)
- Fixed bug in range optimizer when using overlapping ranges. (Bug #2448)
- Limit `wait_timeout` [434] to 2147483 on Windows (OS limit). (Bug #2400)
- Fixed bug when `--init-file` [387] crashes MySQL if it contains a large `SELECT`. (Bug #2526)
- `SHOW KEYS` now shows `NULL` in the `Sub_part` column for `FULLTEXT` indexes.
- The signal thread's stack size was increased to enable `mysqld` to run on Debian/IA-64 with a TLS-enabled `glibc`. (Bug #2599)
- Now only the `SELECT` [492] privilege is needed for tables that are only read in multiple-table `UPDATE` statements. (Bug #2377)
- Give proper error message if one uses `LOCK TABLES ... ; INSERT ... SELECT` and one used the same table in the `INSERT` and `SELECT` part. (Bug #2296)
- `SELECT INTO ... DUMPFILE` now deletes the generated file on error.
- Fixed foreign key reference handling to allow references to column names that contain spaces. (Bug #1725)

- Fixed problem with index reads on character columns with `BDB` tables. The symptom was that data could be returned in the wrong lettercase. (Bug #2509)
- Fixed a spurious table corruption problem that could sometimes appear on tables with indexed `TEXT` columns if these columns happened to contain values having trailing spaces. This bug was introduced in 4.0.17.
- Fixed a problem where some queries could hang if a condition like *`indexed_TEXT_column = expr`* was present and the column contained values having trailing spaces. This bug was introduced in 4.0.17.
- Fixed a bug that could cause incorrect results from a query that involved range conditions on indexed `TEXT` columns that happened to contain values having trailing spaces. This bug was introduced in 4.0.17. (Bug #2295)
- Fixed incorrect path names in some of the manual pages. (Bug #2270)
- Fixed spurious “table corrupted” errors in parallel repair operations. See Section 5.1.3, “Server System Variables”.
- Fixed a crashing bug in parallel repair operations. See Section 5.1.3, “Server System Variables”.
- Fixed bug in updating `MyISAM` tables for `BLOB` values longer than 16MB. (Bug #2159)
- Fixed bug in `mysqld_safe` when running multiple instances of MySQL. (Bug #2114)
- Fixed a bug in using `HANDLER` statement with tables not from a current database. (Bug #2304)
- Fixed a crashing bug that occurred due to the fact that multiple-table `UPDATE` statements did not check that there was only one table to be updated. (Bug #2103)
- Fixed a crashing bug that occurred due to `BLOB` data type index size being calculated incorrectly in `MIN()` [884] and `MAX()` [884] optimizations. (Bug #2189)
- Fixed a bug with incorrect syntax for `LOCK TABLES` in `mysqldump`. (Bug #2242)
- Fixed a bug in `mysqld_safe` that caused `mysqld` to generate a warning about duplicate `user=`*`xxx`* options if this option was specified in the `[mysqld]` or `[server]` sections of `my.cnf`. (Bug #2163)
- `INSERT DELAYED ... SELECT ...` could cause table corruption because tables were not locked properly. This is now fixed by ignoring `DELAYED` in this context. (Bug #1983)
- Replication: Sometimes the master gets a nonfatal error during the execution of a statement that does not immediately succeed. (For example, a write to a `MyISAM` table may first receive “no space left on device,” but later complete when disk space becomes available. See Section B.5.4.3, “How MySQL Handles a Full Disk”.) The bug was that the master forgot to reset the error code to 0 after success, so the error code got into its binary log, thus causing the slave to issue false alarms such as “did not get the same error as on master.” (Bug #2083)
- Removed a misleading “check permissions on master.info” from a replication error message, because the cause of the problem could be something other than permissions. (Bug #2121)
- Fixed a crash when the replication slave was unable to create the first relay log. (Bug #2145)
- Replication of `LOAD DATA INFILE` for an empty file from a 3.23 master to a 4.0 slave caused the slave to print an error. (Bug #2452)
- When automatically forcing `lower_case_table_names` [418] to 1 if the file system was case insensitive, `mysqld` could crash. This bug existed only in MySQL 4.0.17. (Bug #2481)

- Restored ability to specify default values for `TIMESTAMP` columns that was erroneously disabled in previous release. (Bug #2539) Fixed `SHOW CREATE TABLE` to reflect these values. (Bug #1885) Note that because of the auto-update feature for the first `TIMESTAMP` column in a table, it makes no sense to specify a default value for the column. Any such default is silently ignored (unless another `TIMESTAMP` column is added before this one). Also fixed the meaning of the `DEFAULT` keyword when it is used to specify the value to be inserted into a `TIMESTAMP` column other than the first. (Bug #2464)
- Fixed bug for out-of-range arguments on QNX platform that caused `UNIX_TIMESTAMP()` [841] to produce incorrect results or that caused nonzero values to be inserted into `TIMESTAMP` columns. (Bug #2523) Also, current time zone now is taken into account when checking if datetime values satisfy both range boundaries for `TIMESTAMP` columns. The range permitted for a `TIMESTAMP` column is time zone-dependent and equivalent to a range of `1970-01-01 00:00:01` UTC to `2037-12-31 23:59:59` UTC.
- Multiple-table `DELETE` statements were never replicated by the slave if there were any `--replicate-*-table` options. (Bug #2527)
- Changes to session counterparts of variables `query_prealloc_size` [425], `query_alloc_block_size` [424], `trans_prealloc_size`, `trans_alloc_block_size` now have an effect. (Bug #1948)
- Fixed bug in `ALTER TABLE RENAME`, when rename to the table with the same name in another database silently dropped destination table if it existed. (Bug #2628)

C.2.15 Changes in Release 4.0.17 (14 December 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- `mysqldump` no longer dumps data for `MERGE` tables. (Bug #1846)
- `lower_case_table_names` [418] is now forced to 1 if the database directory is located on a case-insensitive file system. (Bug #1812)
- Symlink creation is now disabled on systems where `realpath()` doesn't work. (Before one could use `CREATE TABLE .. DATA DIRECTORY=..` even if `HAVE_BROKEN_REALPATH` was defined. This is now disabled to avoid problems when running `ALTER TABLE`).
- Inserting a negative `AUTO_INCREMENT` value in a `MyISAM` table no longer updates the `AUTO_INCREMENT` counter to a big unsigned value. (Bug #1366)
- Added four new modes to `WEEK(..., mode)` [843] function. (Bug #1178)
- Permit `UNION DISTINCT` syntax.
- MySQL now syncs to disk each time `.frm` file is created.
- `mysql_server_init()` now returns 1 if it can't initialize the environment. (Previously `mysql_server_init()` called `exit(1)` if it could not create a key with `pthread_key_create()`. (Bug #2062)
- Permit spaces in Windows service names.

- Changed the default Windows service name for `mysqld` from `MySql` to `MySQL`. This should not affect usage, because service names are not case sensitive.
- When you install `mysqld` as a service on Windows systems, `mysqld` reads startup options in option files from the option group with the same name as the service name. (Except when the service name is `MySQL`).

Bugs fixed:

- Sending `SIGHUP` to `mysqld` crashed the server if it was running with `--log-bin` [1181]. (Bug #2045)
- One can now configure MySQL as a Windows service as a normal user. (Bug #1802). Thanks to Richard Hansen for fixing this.
- Database names are now compared in lowercase in `ON` clauses when `lower_case_table_names` [418] is set. (Bug #1736)
- `IGNORE ... LINES` option to `LOAD DATA INFILE` didn't work when used with fixed length rows. (Bug #1704)
- Fixed problem with `UNIX_TIMESTAMP()` [841] for timestamps close to 0. (Bug #1998)
- Fixed problem with character values greater than 128 in the `QUOTE()` [800] function. (Bug #1868)
- Fixed searching of `TEXT` with endspace. (Bug #1651)
- Fixed caching bug in multiple-table updates where same table was used twice. (Bug #1711)
- Fixed directory permissions for the MySQL-server RPM documentation directory. (Bug #1672)
- Fixed server crash when updating an `ENUM` column that is set to the empty string (for example, with `REPLACE()` [800]). (Bug #2023)
- `mysql` client program now correctly prints connection identifier returned by `mysql_thread_id()` as unsigned integer rather than as signed integer. (Bug #1951)
- `FOUND_ROWS()` [872] could return incorrect number of rows after a query with an impossible `WHERE` condition. (Bug #1468)
- `SHOW DATABASES` no longer shows `.sym` files (on Windows) that do not point to a valid directory. (Bug #1385)
- Fixed a possible memory leak on Mac OS X when using the shared `libmysql.so` library. (from `pthread_key_create()`). (Bug #2061)
- Fixed bug in `UNION` statement with alias `*`. (Bug #1249)
- Fixed a bug in `DELETE ... ORDER BY ... LIMIT` where the rows were not deleted in the proper order. (Bug #1024, Bug #1697).
- Fixed serious problem with multi-threaded programs on Windows that used the embedded MySQL libraries. (Locks of tables were not handled correctly between different threads).
- Code cleanup: Fixed a few code defects (potential memory leaks, null pointer dereferences, uninitialized variables). Thanks to Reasoning Inc. for informing us about these findings.
- Fixed a buffer overflow error that occurred with prepended "`0`" characters in some columns of type `DECIMAL`. (Bug #2128)
- Filesort was never shown in `EXPLAIN` if query contained an `ORDER BY NULL` clause. (Bug #1335)

- Fixed invalidation of whole query cache on `DROP DATABASE`. (Bug #1898)
- Fixed bug in range optimizer that caused wrong results for some unlikely `AND` [787]/`OR` [787] queries. (Bug #1828)
- Fixed a crash in `ORDER BY` when ordering by expression and identifier. (Bug #1945)
- Fixed a crash in an open `HANDLER` when an `ALTER TABLE` was executed in a different connection. (Bug #1826)
- Fixed a bug in `trunc*` operator of full-text search which sometimes caused MySQL not to find all matched rows.
- Fixed bug in prepending "`0`" characters to `DECIMAL` column values.
- Fixed optimizer bug, introduced in 4.0.16, when `REF` access plan was preferred to more efficient `RANGE` on another column.
- Fixed problem when installing a MySQL server as a Windows service using a command of the form `mysqld --install mysql --defaults-file=`*`path-to-file`*`.` (Bug #1643)
- Fixed an incorrect result from a query that uses only `const` [567] tables (such as one-row tables) and nonconstant expression (such as `RAND()` [822]). (Bug #1271)
- Fixed bug when the optimizer did not take `SQL_CALC_FOUND_ROWS` into account if `LIMIT` clause was present. (Bug #1274)
- `mysqlbinlog` now asks for a password at the console when the `-p` or `--password` [341] option is used with no argument. This is consistent with the way that other clients such `mysqladmin` and `mysqldump` behave.

  Note

  A consequence of this change is that it is no longer possible to invoke `mysqlbinlog` as `mysqlbinlog -p pass_val` (with a space between the `-p` option and the following password value). (Bug #1595)
- Fixed bug accidentally introduced in 4.0.16 where the slave SQL thread deleted its replicated temporary tables when `STOP SLAVE` was issued.
- In a "chain" replication setup `A->B->C`, if 2 sessions on A updated temporary tables of the same name at the same time, the binary log of `B` became incorrect, resulting in `C` becoming confused. (Bug #1686)
- In a "chain" replication setup `A->B->C`, if `STOP SLAVE` was issued on `B` while it was replicating a temporary table from `A`, then when `START SLAVE` was issued on `B`, the binary log of `B` became incorrect, resulting in `C` becoming confused. (Bug #1240)
- When `MASTER_LOG_FILE` and `MASTER_LOG_POS` were not specified, `CHANGE MASTER TO` used the coordinates of the slave I/O thread to set up replication, which broke replication if the slave SQL thread lagged behind the slave I/O thread. This caused the slave SQL thread to lose some events. The new behavior is to use the coordinates of the slave SQL thread instead. See Section 12.5.2.1, "`CHANGE MASTER TO` Syntax". (Bug #1870)
- Now if integer is stored or converted to `TIMESTAMP` or `DATETIME` value checks of year, month, day, hour, minute and second ranges are performed and numbers representing illegal timestamps are converted to 0 value. This behavior is consistent with manual and with behavior of string to `TIMESTAMP`/`DATETIME` conversion. (Bug #1448)

- Fixed bug when `BIT_AND()` [882] and `BIT_OR()` [882] group functions returned incorrect value if `SELECT` used a temporary table and no rows were found. (Bug #1790).
- `BIT_AND()` [882] is now unsigned in all contexts. This means that it now returns 18446744073709551615 (= 0xffffffffffffffff) instead of -1 if there were no rows in the result.
- Fixed bug with `BIT_AND()` [882] still returning signed value for an empty set in some cases. (Bug #1972)
- Fixed bug with `^` [863] (XOR) and `>>` [863] (bit shift) still returning signed value in some cases. (Bug #1993)
- Replication: a rare race condition in the slave SQL thread, which could lead to a wrong complain that the relay log is corrupted. (Bug #2011)
- Replication: in the slave SQL thread, a multiple-table `UPDATE` could produce a wrong complain that some record was not found in one table, if the `UPDATE` was preceded by a `INSERT ... SELECT`. (Bug #1701)
- Fixed deficiency in MySQL code which is responsible for scanning directories. This deficiency caused `SHOW TABLE STATUS` to be very slow when a database contained a large number of tables, even if a single particular table were specified. (Bug #1952)

C.2.16 Changes in Release 4.0.16 (17 October 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Option values in option files now may be quoted. This is useful for values that contain whitespace or comment characters.
- Write memory allocation information to error log when doing `mysqladmin debug`. This works only on systems that support the `mallinfo()` call (like newer Linux systems).
- Added the following new system variables to allow more precise memory allocation: `range_alloc_block_size` [425], `query_alloc_block_size` [424], `query_prealloc_size` [425], `transaction_alloc_block_size` [432], and `transaction_prealloc_size` [432].
- `mysqlbinlog` now reads option files. To make this work, you must now specify `--read-from-remote-server` [341] when reading binary logs from a MySQL server. (Note that using a remote server is deprecated and may disappear in future `mysqlbinlog` versions).
- Block `SIGPIPE` signals also for nonthreaded programs. The blocking is moved from `mysql_init()` to `mysql_server_init()`, which is automatically called on the first call to `mysql_init()`.
- Added `--libs_r` [357] and `--include` [357] options to `mysql_config`.
- New `` `> `` prompt for `mysql`. This prompt is similar to the `'>` and `">` prompts, but indicates that an identifier quoted with backticks was begun on an earlier line and the closing backtick has not yet been seen.
- Updated `mysql_install_db` to be able to use the local machine's IP address instead of the host name when building the initial grant tables if `skip-name-resolve` has been specified. This option can be

helpful on FreeBSD to avoid thread-safety problems with the FreeBSD resolver libraries. (Thanks to Jeremy Zawodny for the patch.)

- A documentation change: Added a note that when backing up a slave, it is necessary also to back up the `master.info` and `relay-log.info` files, as well as any `SQL_LOAD-*` files located in the directory specified by the `--slave-load-tmpdir` [1179] option. All these files are needed when the slave resumes replication after you restore the slave's data.

Bugs fixed:

- Fixed a spurious error `ERROR 14: Can't change size of file (Errcode: 2)` on Windows in `DELETE FROM tbl_name` without a `WHERE` clause or `TRUNCATE TABLE tbl_name`, when `tbl_name` is a `MyISAM` table. (Bug #1397)
- Fixed a bug that resulted in `thr_alarm queue is full` warnings after increasing the `max_connections` [419] variable with `SET GLOBAL`. (Bug #1435)
- Made `LOCK TABLES` to work when `Lock_tables_priv` is granted on the database level and `Select_priv` is granted on the table level.
- Fixed crash of `FLUSH QUERY CACHE` on queries that use same table several times (Bug #988).
- Fixed core dump bug when setting an enum system variable (such as `sql_warnings` [430]) to `NULL`.
- Extended the default timeout value for Windows clients from 30 seconds to 1 year. (The timeout that was added in MySQL 4.0.15 was way too short). This fixes a bug that caused `ERROR 2013: Lost connection to MySQL server during query` for queries that lasted longer than 30 seconds, if the client didn't specify a limit with `mysql_options()`. Users of 4.0.15 on Windows should upgrade to avoid this problem.
- More “out of memory” checking in range optimizer.
- Fixed and documented a problem when setting and using a user variable within the same `SELECT` statement. (Bug #1194).
- Fixed bug in overrun check for `BLOB` values with compressed tables. This was a bug introduced in 4.0.14. It caused MySQL to regard some correct tables containing `BLOB` values as corrupted. (Bug #770, Bug #1304, and maybe Bug #1295)
- `SHOW GRANTS` showed `USAGE` [493] instead of the real column-level privileges when no table-level privileges were given.
- When copying a database from the master, `LOAD DATA FROM MASTER` dropped the corresponding database on the slave, thus erroneously dropping tables that had no counterpart on the master and tables that may have been excluded from replication using `--replicate-*-table` rules. Now `LOAD DATA FROM MASTER` no longer drops the database. Instead, it drops only the tables that have a counterpart on the master and that match the `--replicate-*-table` rules. `--replicate-*-db` rules can still be used to include or exclude a database as a whole from `LOAD DATA FROM MASTER`. A database also is included or excluded as a whole if there are some rules like `--replicate-wild-do-table=db1.%` [1178] or `--replicate-wild-ignore-table=db1.%` [1178], as is the case for `CREATE DATABASE` and `DROP DATABASE` in replication. (Bug #1248)
- Fixed a bug where `mysqlbinlog` crashed with a segmentation fault when used with the `-h` or `--host` [340] option. (Bug #1258)
- Fixed a bug where `mysqlbinlog` crashed with a segmentation fault when used on a binary log containing only final events for `LOAD DATA`. (Bug #1340)

- `mysqlbinlog` does not reuse temporary file names from previous runs. Previously `mysqlbinlog` failed if was used several times on the same binary log file that contained a `LOAD DATA` statement.
- Fixed compilation problem when compiling with OpenSSL 0.9.7 with disabled old DES support (If `OPENSSL_DISABLE_OLD_DES_SUPPORT` option was enabled).
- Fixed a bug when two (or more) MySQL servers were running on the same machine, and they were both slaves, and at least one of them was replicating some `LOAD DATA INFILE` statement from its master. The bug was that one slave MySQL server sometimes deleted the `SQL_LOAD-*` files (used for replication of `LOAD DATA INFILE` and located in the `slave-load-tmpdir` directory, which defaults to `tmpdir` [432]) belonging to the other slave MySQL server of this machine, if these slaves had the same `slave-load-tmpdir` directory. When that happened, the other slave could not replicate `LOAD DATA INFILE` and complained about not being able to open some `SQL_LOAD-*` file. (Bug #1357)
- If `LOAD DATA INFILE` failed for a small file, the master forgot to write a marker (a `Delete_file` event) in its binary log, so the slave could not delete 2 files (`SQL_LOAD-*.info` and `SQL_LOAD-*.data` from its `tmpdir` [432]. (Bug #1391)
- On Windows, the slave forgot to delete a `SQL_LOAD-*.info` file from `tmpdir` [432] after successfully replicating a `LOAD DATA INFILE` statement. (Bug #1392)
- When a connection terminates, MySQL writes `DROP TEMPORARY TABLE` statements to the binary log for all temporary tables which the connection had not explicitly dropped. MySQL forgot to use backticks to quote the database and table names in the statement. (Bug #1345)
- On some 64-bit machines (some HP-UX and Solaris machines), a slave installed with the 64-bit MySQL binary could not connect to its master (it connected to itself instead). (Bug #1256, Bug #1381)
- Code was introduced in MySQL 4.0.15 for the slave to detect that the master had died while writing a transaction to its binary log. This code reported an error in a legal situation: When the slave I/O thread was stopped while copying a transaction to the relay log, the slave SQL thread would later pretend that it found an unfinished transaction. (Bug #1475)

C.2.17 Changes in Release 4.0.15 (03 September 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Important

If you are using this release on Windows, you should upgrade at least your clients (any program that uses `libmysql.lib`) to 4.0.16 or above. This is because the 4.0.15 release had a bug in the Windows client library that causes Windows clients using the library to die with a `Lost connection to MySQL server during query` error for queries that take more than 30 seconds. This problem is specific to Windows; clients on other platforms are unaffected.

Functionality added or changed:

- `mysqldump` now correctly quotes all identifiers when communicating with the server. This assures that during the dump process, `mysqldump` never sends queries to the server that result in a syntax error. This problem is **not** related to the `mysqldump` program's output, which was not changed. (Bug #1148)
- Change result set metadata information so that `MIN()` [884] and `MAX()` [884] report that they can return `NULL` (this is true because an empty set returns `NULL`). (Bug #324)

- Produce an error message on Windows if a second `mysqld` server is started on the same TCP/IP port as a running `mysqld` server.
- The `mysqld` system variables `wait_timeout` [434], `net_read_timeout` [422], and `net_write_timeout` [422] now work on Windows. One can now also set timeouts for read and writes in Windows clients with `mysql_options()`.
- Added option `--sql-mode=NO_DIR_IN_CREATE` [394] to make it possible for slaves to ignore `INDEX DIRECTORY` and `DATA DIRECTORY` options given to `CREATE TABLE`. When this is mode is on, `SHOW CREATE TABLE` does not show the given directories.
- `SHOW CREATE TABLE` now shows the `INDEX DIRECTORY` and `DATA DIRECTORY` options, if they were specified when the table was created.
- The `open_files_limit` [423] system variable now shows the real open files limit.
- `MATCH ... AGAINST()` in natural language mode now treats words that are present in more than 2,000,000 rows as stopwords.
- The Mac OS X installation disk images now include an additional `MySQLStartupItem.pkg` package that enables the automatic startup of MySQL on system startup. See Section 2.5, “Installing MySQL on Mac OS X”.
- Most of the documentation included in the binary tarball distributions (`.tar.gz`) has been moved into a subdirectory `docs`. See Section 2.1.5, “Installation Layouts”.
- The manual is now included as an additional `info` file in the binary distributions. (Bug #1019)
- The binary distributions now include the embedded server library (`libmysqld.a`) by default. Due to a linking problem with non-`gcc` compilers, it was not included in all packages of the initial 4.0.15 release. The affected packages were rebuilt and released as 4.0.15a. See Section 1.5, “MySQL 4.0 in a Nutshell”.
- MySQL can now use range optimization for `BETWEEN` with nonconstant limits. (Bug #991)
- Replication error messages now include the default database, so that users can check which database the failing query was run for.
- A documentation change: Added a paragraph about how the `binlog-do-db` and `binlog-ignore-db` options are tested against the database on the master (see Section 5.3.4, “The Binary Log”), and a paragraph about how `--replicate-do-db` [1176], `--replicate-do-table` [1177] and analogous options are tested against the database and tables on the slave (see Section 14.8, “Replication and Binary Logging Options and Variables”).
- Now the slave does not replicate `SET PASSWORD` if it is configured to exclude the `mysql` database from replication (using for example `--replicate-wild-ignore-table=mysql.%` [1178]). This was the case for `GRANT` and `REVOKE` since version 4.0.13 (although there was Bug #980 in 4.0.13 & 4.0.14, which has been fixed in 4.0.15).
- Rewrote the information shown in the `State` column of `SHOW PROCESSLIST` for replication threads and for `MASTER_POS_WAIT()` [879] and added the most common states for these threads to the documentation, see Section 14.3, “Replication Implementation Details”.
- Added a test in replication to detect the case where the master died in the middle of writing a transaction to the binary log; such unfinished transactions now trigger an error message on the slave.
- A `GRANT` statement that creates an anonymous user (that is, an account with an empty user name) no longer requires `FLUSH PRIVILEGES` for the account to be recognized by the server. (Bug #473)

- `CHANGE MASTER TO` now flushes `relay-log.info`. Previously this was deferred to the next run of `START SLAVE`, so if `mysqld` was shutdown on the slave after `CHANGE MASTER TO` without having run `START SLAVE`, the relay log's name and position were lost. At restart they were reloaded from `relay-log.info`, thus reverting to their old (incorrect) values from before `CHANGE MASTER TO` and leading to error messages (as the old relay log did not exist any more) and the slave threads refusing to start. (Bug #858)

Bugs fixed:

- Fixed buffer overflow in password handling which could potentially be exploited by MySQL users with `ALTER` [491] privilege on the `mysql.user` table to execute random code or to gain shell access with the UID of the `mysqld` process (thanks to Jedi/Sector One for spotting and reporting this bug). (CVE-2003-0780)
- Fixed server crash on `FORCE INDEX` in a query that contained "Range checked for each record" in the `EXPLAIN` output. (Bug #1172)
- Fixed table/column grant handling: The proper sort order (from most specific to less specific, see Section 5.5.5, “Access Control, Stage 2: Request Verification”) was not honored. (Bug #928)
- Fixed rare bug in `MYISAM` introduced in 4.0.3 where the index file header was not updated directly after an `UPDATE` of split dynamic rows. The symptom was that the table had a corrupted delete-link if `mysqld` was shut down or the table was checked directly after the update.
- Fixed `Can't unlock file` error when running `myisamchk --sort-index` on Windows. (Bug #1119)
- Fixed possible deadlock when changing `key_buffer_size` [415] while the key cache was actively used. (Bug #1088)
- Fixed overflow bug in `MyISAM` and `ISAM` when a row is updated in a table with a large number of columns and at least one `BLOB/TEXT` column.
- Fixed incorrect result when doing `UNION` and `LIMIT #,#` when braces were not used around the `SELECT` parts.
- Fixed incorrect result when doing `UNION` and `ORDER BY .. LIMIT #` when one didn't use braces around the `SELECT` parts.
- Fixed problem with `SELECT SQL_CALC_FOUND_ROWS ... UNION ALL ... LIMIT #` where `FOUND_ROWS()` [872] returned incorrect number of rows.
- Fixed unlikely stack bug when having a BIG expression of type `1+1-1+1-1...` in certain combinations. (Bug #871)
- Fixed the bug that sometimes prevented a table with a `FULLTEXT` index from being marked as "analyzed".
- Fixed MySQL so that the column length (in C API) for the second column in `SHOW CREATE TABLE` is always larger than the data length. The only known application that was affected by the old behavior was Borland dbExpress, which truncated the output from the command. (Bug #1064)
- Fixed crash in comparisons of strings using the `tis620` character set. (Bug #1116)
- Fixed `ISAM` bug in `MAX()` [884] optimization.
- `myisamchk --sort-records=N` no longer marks table as crashed if sorting failed because of an inappropriate key. (Bug #892)

- Fixed a minor bug in `MyISAM` compressed table handling that sometimes made it impossible to repair compressed table in "Repair by sort" mode. "Repair with keycache" (`myisamchk --safe-recover`) worked, though. (Bug #1015)
- Fixed bug in propagating the version number to the manual included in the distribution files. (Bug #1020)
- Fixed key sorting problem (a `PRIMARY` key declared for a column that is not explicitly marked `NOT NULL` was sorted after a `UNIQUE` key for a `NOT NULL` column).
- Fixed the result of `INTERVAL` when applied to a `DATE` value. (Bug #792)
- Fixed compiling of the embedded server library in the RPM spec file. (Bug #959)
- Added some missing files to the RPM spec file and fixed some RPM building errors that occurred on Red Hat Linux 9. (Bug #998)
- Fixed incorrect `XOR` evaluation in `WHERE` clause. (Bug #992)
- Fixed bug with processing in query cache merged tables constructed from more than 255 tables. (Bug #930)
- Fixed incorrect results from outer join query (for example, `LEFT JOIN`) when `ON` condition is always false, and range search in used. (Bug #926)
- Fixed a bug causing incorrect results from `MATCH ... AGAINST()` in some joins. (Bug #942)
- `MERGE` tables do not ignore `Using index` (from `EXPLAIN` output) anymore.
- Fixed a bug that prevented an empty table from being marked as "analyzed". (Bug #937)
- Fixed `myisamchk --sort-records` crash when used on compressed table.
- Fixed slow (as compared to 3.23) `ALTER TABLE` and related commands such as `CREATE INDEX`. (Bug #712)
- Fixed segmentation fault resulting from `LOAD DATA FROM MASTER` when the master was running without the `--log-bin` [1181] option. (Bug #934)
- Fixed a security bug: A server compiled without SSL support still permitted connections by users who had the `REQUIRE SSL` option specified for their accounts.
- Fixed a random bug: Sometimes the slave would replicate `GRANT` or `REVOKE` queries even if it was configured to exclude the `mysql` database from replication (for example, using `--replicate-wild-ignore-table=mysql.%` [1178]). (Bug #980)
- The `Last_Errno` and `Last_Error` fields in the output of `SHOW SLAVE STATUS` are now cleared by `CHANGE MASTER TO` and when the slave SQL thread starts. (Bug #986)
- A documentation mistake: It said that `RESET SLAVE` does not change connection information (master host, port, user, and password), whereas it does. The statement resets these to the startup options (`master-host` etc) if there were some. (Bug #985)
- `SHOW SLAVE STATUS` now shows correct information (master host, port, user, and password) after `RESET SLAVE` (that is, it shows the new values, which are copied from the startup options if there were some). (Bug #985)
- Disabled propagation of the original master's log position for events because this caused unexpected values for `Exec_Master_Log_Pos` and problems with `MASTER_POS_WAIT()` [879] in A->B->C replication setup. (Bug #1086)

- Fixed a segmentation fault in `mysqlbinlog` when `--position=x` [341] was used with `x` being between a `Create_file` event and its fellow `Append_block`, `Exec_load` or `Delete_file` events. (Bug #1091)
- `mysqlbinlog` printed superfluous warnings when using `--database` [339], which caused syntax errors when piped to `mysql`. (Bug #1092)
- Made `mysqlbinlog --database` filter `LOAD DATA INFILE` too (previously, it filtered all queries except `LOAD DATA INFILE`). (Bug #1093)
- `mysqlbinlog` in some cases forgot to put a leading `'#'` in front of the original `LOAD DATA INFILE` (this command is displayed only for information, not to be run; it is later reworked to `LOAD DATA LOCAL` with a different file name, for execution by `mysql`). (Bug #1096)
- `binlog-do-db` and `binlog-ignore-db` incorrectly filtered `LOAD DATA INFILE` (it was half-written to the binary log). This resulted in a corrupted binary log, which could cause the slave to stop with an error. (Bug #1100)
- When, in a transaction, a transactional table (such as an `InnoDB` table) was updated, and later in the same transaction a nontransactional table (such as a `MyISAM` table) was updated using the updated content of the transactional table (with `INSERT ... SELECT` for example), the queries were written to the binary log in an incorrect order. (Bug #873)
- When, in a transaction, `INSERT ... SELECT` updated a nontransactional table, and `ROLLBACK` was issued, no error was returned to the client. Now the client is warned that some changes could not be rolled back, as this was the case for normal `INSERT`. (Bug #1113)
- Fixed a potential bug: When `STOP SLAVE` was run while the slave SQL thread was in the middle of a transaction, and then `CHANGE MASTER TO` was used to point the slave to some nontransactional statement, the slave SQL thread could get confused (because it would still think, from the past, that it was in a transaction).

C.2.18 Changes in Release 4.0.14 (18 July 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Added `default_week_format` [410] system variable. The value is used as the default mode for the `WEEK()` [843] function.
- `mysqld` now reads an additional option file group having a name corresponding to the server's release series: `[mysqld-4.0]` for 4.0.x servers, `[mysqld-4.1]` for 4.1.x servers, and so forth. This enables options to be specified on a series-specific basis.
- The `CONCAT_WS()` [795] function no longer skips empty strings. (Bug #586).
- `InnoDB` now supports indexing a prefix of a column. This means, in particular, that `BLOB` and `TEXT` columns can be indexed in `InnoDB` tables, which was not possible before.
- A documentation change: Function `INTERVAL(NULL, ...)` [785] returns `-1`.
- Enabled `INSERT` from `SELECT` when the table into which the records are inserted is also a table listed in the `SELECT`.

- Permit `CREATE TABLE` and `INSERT` from any `UNION`.
- The `SQL_CALC_FOUND_ROWS` option now always returns the total number of rows for any `UNION`.
- Removed `--table` option from `mysqlbinlog` to avoid repeating `mysqldump` functionality.
- Comment lines in option files can now start from the middle of a line, too (like `basedir=c:\mysql # installation directory`).
- Changed optimizer slightly to prefer index lookups over full table scans in some boundary cases.
- Added thread-specific `max_seeks_for_key` [420] variable that can be used to force the optimizer to use keys instead of table scans even if the cardinality of the index is low.
- Added optimization that converts `LEFT JOIN` to normal join in some cases.
- A documentation change: added a paragraph about failover in replication (how to use a surviving slave as the new master, how to resume to the original setup). See Section 14.10, “Replication FAQ”.
- A documentation change: added warning notes about safe use of the `CHANGE MASTER TO` statement. See Section 12.5.2.1, “`CHANGE MASTER TO` Syntax”.
- MySQL now issues a warning (not an error, as in 4.0.13) when it opens a table that was created with MySQL 4.1.
- Added `--nice` [245] option to `mysqld_safe` to allow setting the niceness of the `mysqld` process. (Thanks to Christian Hammers for providing the initial patch.) (Bug #627)
- Added `--read-only` [1174] option to cause `mysqld` to allow no updates except from slave threads or from users with the `SUPER` [493] privilege. (Original patch from Markus Benning).
- `SHOW BINLOG EVENTS FROM x` where `x` is less than 4 now silently converts `x` to 4 instead of printing an error. The same change was done for `CHANGE MASTER TO MASTER_LOG_POS=x` and `CHANGE MASTER TO RELAY_LOG_POS=x`.
- `mysqld` now only adds an interrupt handler for the `SIGINT` signal if you start it with the new `--gdb` [387] option. This is done because some MySQL users encountered strange problems when they accidentally sent `SIGINT` to `mysqld` threads.
- `RESET SLAVE` now clears the `Last_Errno` and `Last_Error` fields in the output of `SHOW SLAVE STATUS`.
- Added `max_relay_log_size` [420] variable; the relay log is rotated automatically when its size exceeds `max_relay_log_size` [420]. But if `max_relay_log_size` [420] is 0 (the default), `max_binlog_size` [1184] is used (as in older versions). `max_binlog_size` [1184] still applies to binary logs in any case.
- `FLUSH LOGS` now rotates relay logs in addition to the other types of logs it rotates.

Bugs fixed:

- Comparison/sorting for `latin1_de` character set was rewritten. The old algorithm could not handle cases like `"sä" > "ßa"`. See Section 9.2, “Using the German Character Set”. In rare cases it resulted in table corruption.
- Fixed a problem with the password prompt on Windows. (Bug #683)
- `ALTER TABLE ... UNION=(...)` for `MERGE` table is now permitted even if some underlying `MyISAM` tables are read only. (Bug #702)

- Fixed a problem with `CREATE TABLE t1 SELECT x'41'`. (Bug #801)
- Removed some incorrect lock warnings from the error log.
- Fixed memory overrun when doing `REPAIR TABLE` on a table with a multiple-part auto_increment key where one part was a packed `CHAR`.
- Fixed a probable race condition in the replication code that could potentially lead to `INSERT` statements not being replicated in the event of a `FLUSH LOGS` command or when the binary log exceeds `max_binlog_size` [1184]. (Bug #791)
- Fixed a crashing bug in `INTERVAL` and `GROUP BY` or `DISTINCT`. (Bug #807)
- Fixed bug in `mysqlhotcopy` so it actually aborts for unsuccessful table copying operations. Fixed another bug so that it succeeds when there are thousands of tables to copy. (Bug #812)
- Fixed problem with `mysqlhotcopy` failing to read options from option files. (Bug #808)
- Fixed bugs in optimizer that sometimes prevented MySQL from using `FULLTEXT` indexes even though it was possible (for example, in `SELECT * FROM t1 WHERE MATCH a,b AGAINST("index") > 0`).
- Fixed a bug with “table is full” in `UNION` operations.
- Fixed a security problem that enabled users with no privileges to obtain information on the list of existing databases by using `SHOW TABLES` and similar commands.
- Fixed a stack problem on UnixWare/OpenUnix.
- Fixed a configuration problem on UnixWare/OpenUNIX and OpenServer.
- Fixed a problem with `max_user_connections` [420].
- `HANDLER` without an index now works properly when a table has deleted rows. (Bug #787)
- Fixed a bug with `LOAD DATA` in `mysqlbinlog`. (Bug #670)
- Fixed that `SET CHARACTER SET DEFAULT` works. (Bug #462)
- Fixed `MERGE` table behavior in `ORDER BY ... DESC` queries. (Bug #515)
- Fixed server crash on `PURGE MASTER LOGS` or `SHOW MASTER LOGS` when the binary log is off. (Bug #733)
- Fixed password-checking problem on Windows. (Bug #464)
- Fixed the bug in comparison of a `DATETIME` column and an integer constant. (Bug #504)
- Fixed remote mode of `mysqlbinlog`. (Bug #672)
- Fixed `ERROR 1105: Unknown error` that occurred for some `SELECT` queries, where a column that was declared as `NOT NULL` was compared with an expression that took `NULL` value.
- Changed timeout in `mysql_real_connect()` to use `poll()` instead of `select()` to work around problem with many open files in the client.
- Fixed incorrect results from `MATCH ... AGAINST` used with a `LEFT JOIN` query.
- The maximum value for system variables was limited to 4294967295 when specified on the command line.

- Fixed a bug that sometimes caused spurious “Access denied” errors in `HANDLER ... READ` statements, when a table is referenced through an alias.
- Fixed a portability problem with `safe_malloc`, which caused MySQL to produce “Freeing wrong aligned pointer” errors on SCO 3.2.
- `ALTER TABLE ... ENABLE/DISABLE KEYS` could cause a core dump when done after an `INSERT DELAYED` statement on the same table.
- Fixed problem with conversion of localtime to GMT where some times resulted in different (but correct) timestamps. Now MySQL should use the smallest possible timestamp value in this case. (Bug #316)
- Very small query cache sizes could crash `mysqld`. (Bug #549)
- Fixed a bug (accidentally introduced by us but present only in version 4.0.13) that made `INSERT ... SELECT` into an `AUTO_INCREMENT` column not replicate well. This bug is in the master, not in the slave. (Bug #490)
- Fixed a bug: When an `INSERT ... SELECT` statement inserted rows into a nontransactional table, but failed at some point (for example, due to a “Duplicate key” error), the query was not written to the binary log. Now it is written to the binary log, with its error code, as all other queries are. About the `slave-skip-errors` option for how to handle partially completed queries in the slave, see Section 14.8, “Replication and Binary Logging Options and Variables”. (Bug #491)
- `SET foreign_key_checks = 0` was not replicated properly. The fix probably will not be backported to 3.23.
- On a slave, `LOAD DATA INFILE` which had no `IGNORE` or `REPLACE` clause on the master, was replicated with `IGNORE`. Although this is not a problem if the master and slave data are identical (a `LOAD` that produces no duplicate conflicts on the master produces none on the slave anyway), which is true in normal operation, it is better for debugging not to silently add the `IGNORE`. That way, you can get an error message on the slave and discover that for some reason, the data on master and slave are different and investigate why. (Bug #571)
- On a slave, `LOAD DATA INFILE` printed an incomplete “Duplicate entry '%-.64s' for key %d'” message (the key name and value were not mentioned) in case of duplicate conflict (which does not happen in normal operation). (Bug #573)
- When using a slave compiled with `--debug`, `CHANGE MASTER TO RELAY_LOG_POS` could cause a debug assertion failure. (Bug #576)
- When doing a `LOCK TABLES WRITE` on an `InnoDB` table, commit could not happen, if the query was not written to the binary log (for example, if `--log-bin` [1181] was not used, or `binlog-ignore-db` was used). (Bug #578)
- If a 3.23 master had open temporary tables that had been replicated to a 4.0 slave, and the binary log got rotated, these temporary tables were immediately dropped by the slave (which caused problems if the master used them subsequently). This bug had been fixed in 4.0.13, but in a manner which caused an unlikely inconvenience: If the 3.23 master died brutally (power failure), without having enough time to automatically write `DROP TABLE` statements to its binary log, then the 4.0.13 slave would not notice the temporary tables have to be dropped, until the slave `mysqld` server is restarted. This minor inconvenience is fixed in 3.23.57 and 4.0.14 (meaning the master must be upgraded to 3.23.57 and the slave to 4.0.14 to remove the inconvenience). (Bug #254)
- If `MASTER_POS_WAIT()` [879] was waiting, and the slave was idle, and the slave SQL thread terminated, `MASTER_POS_WAIT()` [879] would wait forever. Now when the slave SQL thread terminates, `MASTER_POS_WAIT()` [879] immediately returns `NULL` (“slave stopped”). (Bug #651)

- After `RESET SLAVE; START SLAVE;`, the `Relay_Log_Space` value displayed by `SHOW SLAVE STATUS` was too big by four bytes. (Bug #763)
- If a query was ignored on the slave (because of `--replicate-ignore-table` [1177] and other similar rules), the slave still checked if the query got the same error code (0, no error) as on the master. So if the master had an error on the query (for example, “Duplicate entry” in a multiple-row insert), then the slave stopped and warned that the error codes didn't match. (Bug #797)

C.2.19 Changes in Release 4.0.13 (16 May 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- `PRIMARY KEY` now implies `NOT NULL`. (Bug #390)
- The Windows binary packages are now compiled with `--enable-local-infile` to match the Unix build configuration.
- Removed timing of tests from `mysql-test-run`. `time` does not accept all required parameters on many platforms (for example, QNX) and timing the tests is not really required (it is not a benchmark anyway).
- `SHOW MASTER STATUS` and `SHOW SLAVE STATUS` required the `SUPER` [493] privilege; now they accept `REPLICATION CLIENT` [492] as well. (Bug #343)
- Added multi-threaded `MyISAM` repair optimization and `myisam_repair_threads` [421] variable to enable it. See Section 5.1.3, “Server System Variables”.
- Added `innodb_max_dirty_pages_pct` [1071] variable which controls amount of dirty pages permitted in `InnoDB` buffer pool.
- `CURRENT_USER()` [872] and `Access denied` error messages now report the host name exactly as it was specified in the `GRANT` statement.
- Removed benchmark results from the source and binary distributions. They are still available in the BK source tree, though.
- `InnoDB` tables now support `ANALYZE TABLE`.
- MySQL now issues an error when it opens a table that was created with MySQL 4.1.
- Option `--new` now changes binary items (`0xFFDF`) to be treated as binary strings instead of numbers by default. This fixes some problems with character sets where it is convenient to input the string as a binary item. After this change you have to convert the binary string to `INTEGER` with a `CAST` if you want to compare two binary items with each other and know which one is bigger than the other. `SELECT CAST(0xfeff AS UNSIGNED) < CAST(0xff AS UNSIGNED)`. This is the default behavior in MySQL 4.1. (Bug #152)
- Enabled `delayed_insert_timeout` [410] on Linux (most modern `glibc` libraries have a fixed `pthread_cond_timedwait()`). (Bug #211)
- Do not create more insert delayed threads than given by `max_delayed_threads` [419]. (Bug #211)
- Changed `UPDATE ... LIMIT` to apply the limit to rows that were matched, whether or not they actually were changed. Previously the limit was applied as a restriction on the number of rows changed.

- Tuned optimizer to favor clustered index over table scan.
- `BIT_AND()` [882] and `BIT_OR()` [882] now return an unsigned 64-bit value.
- Added warnings to error log indicating why a secure connection failed (when running with `--log-warnings` [389]).
- Deprecated the options `--skip-symlink` [393] and `--use-symbolic-links` [393] and replaced them with `--symbolic-links` [393].
- The default option for `innodb_flush_log_at_trx_commit` [1068] was changed from 0 to 1 to make `InnoDB` tables ACID by default. See Section 13.2.4, “`InnoDB` Startup Options and System Variables”.
- Added a feature to `SHOW KEYS` to display keys that are disabled by `ALTER TABLE DISABLE KEYS` statement.
- When using a nonexistent table type with `CREATE TABLE`, first try if the default table type exists before falling back to `MyISAM`.
- Added `MEMORY` as an alias for `HEAP`.
- Renamed function `rnd` to `my_rnd` as the name was too generic and is an exported symbol in `libmysqlclient` (thanks to Dennis Haney for the initial patch).
- Portability fix: renamed `include/dbug.h` to `include/my_dbug.h`.
- `mysqldump` no longer silently deletes the binary logs when invoked with the `--master-data` [296] or `--first-slave` [294] option; while this behavior was convenient for some users, others may suffer from it. Now you must explicitly ask for binary logs to be deleted by using the new `--delete-master-logs` [294] option.
- If the slave is configured (using for example `--replicate-wild-ignore-table=mysql.%` [1178]) to exclude `mysql.user`, `mysql.host`, `mysql.db`, `mysql.tables_priv` and `mysql.columns_priv` from replication, then `GRANT` and `REVOKE` are not replicated.

Bugs fixed:

- Logged `Access denied` error message had incorrect `Using password` value. (Bug #398)
- Fixed bug with `NATURAL LEFT JOIN`, `NATURAL RIGHT JOIN` and `RIGHT JOIN` when using many joined tables. The problem was that the `JOIN` method was not always associated with the tables surrounding the `JOIN` method. If you have a query that uses many `RIGHT JOIN` or `NATURAL ... JOINS` you should verify that they work as you expected after upgrading MySQL to this version. (Bug #291)
- Fixed `mysql` parser not to erroneously interpret “`'`” or “`"`” characters within `/* ... */` comment as beginning a quoted string.
- `mysql` command-line client no longer looks for `\*` commands inside backtick-quoted strings.
- Fixed `Unknown error` when using `UPDATE ... LIMIT`. (Bug #373)
- Fixed problem with ANSI mode and `GROUP BY` with constants. (Bug #387)
- Fixed bug with `UNION` and `OUTER JOIN`. (Bug #386)
- Fixed bug if one used a multiple-table `UPDATE` and the query required a temporary table bigger than `tmp_table_size` [432]. (Bug #286)

- Run `mysql_install_db` with the `-IN-RPM` option for the Mac OS X installation to not fail on systems with improperly configured host name configurations.
- `LOAD DATA INFILE` now reads `000000` as a zero date instead of `"2000-00-00"`.
- Fixed bug that caused `DELETE FROM table WHERE const_expression` always to delete the whole table (even if expression result was false). (Bug #355)
- Fixed core dump bug when using `FORMAT('nan',#)` [796]. (Bug #284)
- Fixed name resolution bug with `HAVING ... COUNT(DISTINCT ...)`.
- Fixed incorrect result from truncation operator (`*`) in `MATCH ... AGAINST()` in some complex joins.
- Fixed a crash in `REPAIR ... USE_FRM` command, when used on a read-only table, nonexistent table, or a table with a crashed index file.
- Fixed a crashing bug in `mysql` monitor program. It occurred if program was started with `--no-defaults` [235], with a prompt that contained the host name and a connection to a nonexistent database was requested.
- Fixed problem when comparing a key for a multi-byte character set. (Bug #152)
- Fixed bug in `LEFT`, `RIGHT` and `MID` when used with multi-byte character sets and some `GROUP BY` queries. (Bug #314)
- Fix problem with `ORDER BY` being discarded for some `DISTINCT` queries. (Bug #275)
- Fixed that `SET sql_big_selects = 1` works as documented (This corrects a new bug introduced in 4.0)
- Fixed some serious bugs in `UPDATE ... ORDER BY`. (Bug #241)
- Fixed unlikely problem in optimizing `WHERE` clause with constant expression like in `WHERE 1 AND (a=1 AND b=1)`.
- Fixed that `SET sql_big_selects = 1` works again.
- Introduced proper backtick quoting for db.table in `SHOW GRANTS`.
- `FULLTEXT` index stopped working after `ALTER TABLE` that converts `TEXT` column to `CHAR`. (Bug #283)
- Fixed a security problem with `SELECT` and wildcarded select list, when user only had partial column `SELECT` [492] privileges on the table.
- Mark a `MyISAM` table as "analyzed" only when all the keys are indeed analyzed.
- Only ignore world-writable `my.cnf` files that are regular files (and not, for example, named pipes or character devices).
- Fixed few smaller issues with `SET PASSWORD`.
- Fixed error message which contained deprecated text.
- Fixed a bug with two `NATURAL JOIN`s in the query.
- `SUM()` [884] didn't return `NULL` when there was no rows in result or when all values was `NULL`.
- On Unix, symbolic link handling was not enabled by default and there was no way to turn this on.

- Added missing dashes to parameter `--open-files-limit` [245] in `mysqld_safe`. (Bug #264)
- Fixed incorrect host name for TCP/IP connections displayed in `SHOW PROCESSLIST`.
- Fixed a bug with `NAN` in `FORMAT(...)` [796] function ...
- Fixed a bug with improperly cached database privileges.
- Fixed a bug in `ALTER TABLE ENABLE / DISABLE KEYS` which failed to force a refresh of table data in the cache.
- Fixed bugs in replication of `LOAD DATA INFILE` for custom parameters (`ENCLOSED`, `TERMINATED` and so on) and temporary tables. (Bug #183, Bug #222)
- Fixed a replication bug when the master is 3.23 and the slave 4.0: the slave lost the replicated temporary tables if `FLUSH LOGS` was issued on the master. (Bug #254)
- Fixed a bug when doing `LOAD DATA INFILE IGNORE`: When reading the binary log, `mysqlbinlog` and the replication code read `REPLACE` instead of `IGNORE`. This could make the slave's table become different from the master's table. (Bug #218)
- Fixed a deadlock when `relay_log_space_limit` [426] was set to a too small value. (Bug #79)
- Fixed a bug in HAVING clause when an alias is used from the **select list**.
- Fixed overflow bug in `MyISAM` when a row is inserted into a table with a large number of columns and at least one `BLOB/TEXT` column. Bug was caused by incorrect calculation of the needed buffer to pack data.
- Fixed a bug when `SELECT` **@non_existent_variable** caused an error in the client/server protocol due to `net_printf()` output being sent to the client twice.
- Fixed a bug in setting the `sql_big_selects` [428] option.
- Fixed a bug in `SHOW PROCESSLIST` which only displayed a localhost in the `"Host"` column. This was caused by a glitch that used only current thread information instead of information from the linked list of threads.
- Removed unnecessary Mac OS X helper files from server RPM. (Bug #144)
- Permit optimization of multiple-table update for `InnoDB` tables as well.
- Fixed a bug in multiple-table updates that caused some rows to be updated several times.
- Fixed a bug in `mysqldump` when it was called with `--master-data` [296]: the `CHANGE MASTER TO` statements appended to the SQL dump had incorrect coordinates. (Bug #159)
- Fixed a bug when an updating query using `USER()` [876] was replicated on the slave; this caused a segmentation fault on the slave. (Bug #178). `USER()` [876] is still badly replicated on the slave (it is replicated to `""`).

C.2.20 Changes in Release 4.0.12 (15 March 2003: Production)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- `mysqld` no longer reads options from world-writable config files. (CVE-2003-0150)
- Integer values between 9223372036854775807 and 9999999999999999999 are now regarded as unsigned longlongs, not as floats. This makes these values work similar to values between 10000000000000000000 and 18446744073709551615.
- `SHOW PROCESSLIST` now includes the client TCP port after the host name to make it easier to know from which client the request originated.
- The `--new` option can be used to make a 4.0 server return `TIMESTAMP` as a string in `'YYYY-MM-DD HH:MM:SS'` format, the way that 4.1 servers do. This is also a `new` [390] system variable that can be set for the same effect. See Section 10.3.1.1, “`TIMESTAMP` Properties Prior to MySQL 4.1”.

Bugs fixed:

- Fixed `mysqld` crash on extremely small values of `sort_buffer` variable.
- `INSERT INTO u SELECT ... FROM t` was written too late to the binary log if t was very frequently updated during the execution of this query. This could cause a problem with `mysqlbinlog` or replication. The master must be upgraded, not the slave. (Bug #136)
- Fixed checking of random part of `WHERE` clause. (Bug #142)
- Fixed a bug with multiple-table updates with `InnoDB` tables. This bug occurred as, in many cases, `InnoDB` tables cannot be updated “on the fly,” but offsets to the records have to be stored in a temporary table.
- Added missing file `mysql_secure_installation` to the `server` RPM subpackage. (Bug #141)
- Fixed MySQL (and `myisamchk`) crash on artificially corrupted `.MYI` files.
- Do not allow `BACKUP TABLE` to overwrite existing files.
- Fixed a bug with multiple-table `UPDATE` statements when user had all privileges on the database where tables are located and there were any entries in `tables_priv` table, that is, `grant_option` was true.
- Fixed a bug that permitted a user with table or column grants on some table, `TRUNCATE TABLE` any table in the same database.
- Fixed deadlock when doing `LOCK TABLE` followed by `DROP TABLE` in the same thread. In this case one could still kill the thread with `KILL`.
- `LOAD DATA LOCAL INFILE` was not properly written to the binary log (hence not properly replicated). (Bug #82)
- `RAND()` [822] entries were not read correctly by `mysqlbinlog` from the binary log which caused problems when restoring a table that was inserted with `RAND()` [822]. `INSERT INTO t1 VALUES(RAND())`. In replication this worked okay.
- `SET sql_log_bin = 0` was ignored for `INSERT DELAYED` queries. (Bug #104)
- `SHOW SLAVE STATUS` reported too old positions (columns `Relay_Master_Log_File` and `Exec_Master_Log_Pos`) for the last executed statement from the master, if this statement was the `COMMIT` of a transaction. The master must be upgraded for that, not the slave. (Bug #52)
- `LOAD DATA INFILE` was not replicated by the slave if `replicate_*_table` was set on the slave. (Bug #86)

- After `RESET SLAVE`, the coordinates displayed by `SHOW SLAVE STATUS` looked un-reset (although they were, but only internally). (Bug #70)
- Fixed query cache invalidation on `LOAD DATA`.
- Fixed memory leak on `ANALYZE` procedure with error.
- Fixed a bug in handling `CHAR(0)` columns that could cause incorrect results from the query.
- Fixed rare bug with incorrect initialization of `AUTO_INCREMENT` column, as a secondary column in a multi-column key (see Section 3.6.9, “Using `AUTO_INCREMENT`”), when data was inserted with `INSERT ... SELECT` or `LOAD DATA` into an empty table.
- On Windows, `STOP SLAVE` didn't stop the slave until the slave got one new command from the master (this bug has been fixed for MySQL 4.0.11 by releasing updated 4.0.11a Windows packages, which include this individual fix on top of the 4.0.11 sources). (Bug #69)
- Fixed a crash when no database was selected and `LOAD DATA` statement was issued with full table name specified, including database prefix.
- Fixed a crash when shutting down replication on some platforms (for example, Mac OS X).
- Fixed a portability bug with `pthread_attr_getstacksize` on HP-UX 10.20 (Patch was also included in 4.0.11a sources).
- Fixed the `bigint` test to not fail on some platforms (for example, HP-UX and Tru64) due to different return values of the `atof()` function.
- Fixed the `rpl_rotate_logs` test to not fail on certain platforms (such as Mac OS X) due to a too-long file name (changed `slave-master-info.opt` to `.slave-mi`).

C.2.21 Changes in Release 4.0.11 (20 February 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- `NULL` is now sorted **LAST** if you use `ORDER BY ... DESC` (as it was before MySQL 4.0.2). This change was required to comply with the SQL standard. (The original change was made because we thought that standard SQL required `NULL` to be always sorted at the same position, but this was incorrect).
- Added `START TRANSACTION` (standard SQL syntax) as alias for `BEGIN`. This is recommended to use instead of `BEGIN` to start a transaction.
- Added `OLD_PASSWORD()` [868] as a synonym for `PASSWORD()` [868].
- Permit keyword `ALL` in group functions.
- Added support for some new `INNER JOIN` and `JOIN` syntaxes. For example, `SELECT * FROM t1 INNER JOIN t2` didn't work before.
- Novell NetWare 6.0 porting effort completed, Novell patches merged into the main source tree.

Bugs fixed:

- Fixed problem with multiple-table delete and `InnoDB` tables.
- Fixed a problem with `BLOB NOT NULL` columns used with `IS NULL` [783].
- Re-added missing pre- and post(un)install scripts to the Linux RPM packages (they were missing after the renaming of the server subpackage).
- Fixed that table locks are not released with multiple-table updates and deletes with `InnoDB` storage engine.
- Fixed bug in updating `BLOB` columns with long strings.
- Fixed integer-wraparound when giving big integer (>= 10 digits) to function that requires an unsigned argument, like `CREATE TABLE (...) AUTO_INCREMENT=N`.
- `MIN(key_column)` [884] could in some cases return `NULL` on a column with `NULL` and other values.
- `MIN(key_column)` [884] and `MAX(key_column)` [884] could in some cases return incorrect values when used in `OUTER JOIN`.
- `MIN(key_column)` [884] and `MAX(key_column)` [884] could return incorrect values if one of the tables was empty.
- Fixed rare crash in compressed `MyISAM` tables with blobs.
- Fixed bug in using aggregate functions as argument for `INTERVAL()` [785], `CASE` [790], `FIELD()` [796], `CONCAT_WS()` [795], `ELT()` [795] and `MAKE_SET()` [799] functions.
- When running with `--lower-case-table-names` [418] (default on Windows) and you had tables or databases with mixed case on disk, then executing `SHOW TABLE STATUS` followed with `DROP DATABASE` or `DROP TABLE` could fail with `Errcode 13`.

C.2.22 Changes in Release 4.0.10 (29 January 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Added option `--log-error[=file_name]` [388] to `mysqld_safe` and `mysqld`. This option forces all error messages to be put in a log file if the option `--console` [385] is not given. On Windows `--log-error` [388] is enabled as default, with a default name of `host_name.err` if the name is not specified.
- Changed some messages from `Warning:` to `Note:` in the log files.
- The `mysqld` server should now compile on NetWare.
- Added optimization that if one does `GROUP BY ... ORDER BY NULL` then result is not sorted.
- New `ft_stopword_file` [412] system variable for `mysqld` to replace/disable the built-in stopword list that is used in full-text searches. See Section 5.1.3, “Server System Variables”.
- Changed default stack size from 64KB to 192KB; This fixes a core dump problem on Red Hat 8.0 and other systems with a `glibc` that requires a stack size larger than 128K for `gethostbyaddr()` to resolve a host name. You can fix this for earlier MySQL versions by starting `mysqld` with `--thread-stack=192K` [431].

- Added `mysql_waitpid` to the binary distribution and the `MySQL-client` RPM subpackage (required for `mysql-test-run`).
- Renamed the main `MySQL` RPM package to `MySQL-server`. When updating from an older version, `MySQL-server.rpm` simply replaces `MySQL.rpm`.
- If a slave is configured with `replicate_wild_do_table=db.%` or `replicate_wild_ignore_table=db.%`, these rules are applied to `CREATE/DROP DATABASE`, too.
- Added timeout value for `MASTER_POS_WAIT()` [879].

Bugs fixed:

- Fixed initialization of the random seed for newly created threads to give a better `rand()` distribution from the first call.
- Fixed a bug that caused `mysqld` to hang when a table was opened with the `HANDLER` statement and then dropped without being closed.
- Fixed bug in logging to binary log (which affects replication) a query that inserts a `NULL` in an `AUTO_INCREMENT` column and also uses `LAST_INSERT_ID()` [874].
- Fixed an unlikely bug that could cause a memory overrun when using `ORDER BY constant_expression`.
- Fixed a table corruption in `myisamchk` parallel repair mode.
- Fixed bug in query cache invalidation on simple table renaming.
- Fixed bug in `mysqladmin --relative`.
- On some 64-bit systems, `show status` reported a strange number for `Open_files` and `Open_streams` [453].
- Fixed incorrect number of columns in `EXPLAIN` on empty table.
- Fixed bug in `LEFT JOIN` that caused zero rows to be returned in the case the `WHERE` condition was evaluated as `FALSE` after reading `const` [567] tables. (Unlikely condition).
- `FLUSH PRIVILEGES` didn't correctly flush table/column privileges when `mysql.tables_priv` is empty.
- Fixed bug in replication when using `LOAD DATA INFILE` one a file that updated an `AUTO_INCREMENT` column with `NULL` or `0`. This bug only affected MySQL 4.0 masters (not slaves or MySQL 3.23 masters).

Note

If you have a slave that has replicated a file with generated `AUTO_INCREMENT` columns, the slave data is corrupted and you should reinitialize the affected tables from the master.

- Fixed possible memory overrun when sending a `BLOB` value larger than 16M to the client.
- Fixed incorrect error message when setting a `NOT NULL` column to an expression that returned `NULL`.
- Fixed core dump bug in *`str`* `LIKE "%`*`other_str`*`%"` where *`str`* or *`other_str`* contained characters >= 128.
- Fixed bug: When executing on master `LOAD DATA` and `InnoDB` failed with `table full` error the binary log was corrupted.

C.2.23 Changes in Release 4.0.9 (09 January 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- `OPTIMIZE TABLE` for `MyISAM` tables treats all `NULL` values as different when calculating cardinality. This helps in optimizing joins between tables where one of the tables has a lot of `NULL` values in a indexed column:

```
SELECT * from t1, t2 where t1.a=t2.key_with_a_lot_of_null;
```

- Added join operator `FORCE INDEX (index_list)`. This acts likes `USE INDEX (index_list)` but with the addition that a table scan is assumed to be VERY expensive. One bad thing with this is that it makes `FORCE` a reserved word.
- Reset internal row buffer in `MyISAM` after each query. This reduces memory in case you have a lot of big blobs in a table.

Bugs fixed:

- A security patch in 4.0.8 causes the `mysqld` server to die if the remote host name can't be resolved. This is now fixed.
- Fixed crash when replication big `LOAD DATA INFILE` statement that caused log rotation.

C.2.24 Changes in Release 4.0.8 (07 January 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Default `max_packet_length` for `libmysqld.c` is now 1024*1024*1024.
- You can now specify `max_allowed_packet` [418] in a file read by `mysql_options(MYSQL_READ_DEFAULT_FILE)`. for clients.
- When sending a too big packet to the server with the not compressed protocol, the client now gets an error message instead of a lost connection.
- We now send big queries/result rows in bigger hunks, which should give a small speed improvement.
- Fixed some bugs with the compressed protocol for rows > 16MB.
- `InnoDB` tables now also support `ON UPDATE CASCADE` in `FOREIGN KEY` constraints. See the `InnoDB` section in the manual for the `InnoDB` changelog.

Bugs fixed:

- Fixed bug in `ALTER TABLE` with `BDB` tables.

- Fixed core dump bug in `QUOTE()` [800] function.
- Fixed a bug in handling communication packets bigger than 16MB. Unfortunately this required a protocol change; If you upgrade the server to 4.0.8 and above and have clients that use packets >= 255*255*255 bytes (=16581375) you must also upgrade your clients to at least 4.0.8. If you don't upgrade, the clients hang when sending a big packet.
- Fixed bug when sending blobs longer than 16MB to client.
- Fixed bug in `GROUP BY` when used on BLOB column with `NULL` values.
- Fixed a bug in handling `NULL` values in `CASE` [790] ... `WHEN` ...

C.2.25 Changes in Release 4.0.7 (20 December 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- `mysqlbug` now also reports the compiler version used for building the binaries (if the compiler supports the option `--version`).

Bugs fixed:

- Fixed compilation problems on OpenUnix and HPUX 10.20.
- Fixed some optimization problems when compiling MySQL with `-DBIG_TABLES` on a 32-bit system.
- `mysql_drop_db()` didn't check permissions properly so anyone could drop another users database. `DROP DATABASE` is checked properly.

C.2.26 Changes in Release 4.0.6 (14 December 2002: Gamma)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Added syntax support for `CHARACTER SET` *`xxx`* and `CHARSET=`*`xxx`* table options (to be able to read table dumps from 4.1).
- Fixed replication bug that caused the slave to loose its position in some cases when the replication log was rotated.
- Fixed that a slave restarts from the start of a transaction if it is killed in the middle of one.
- Moved the manual pages from `man` to `man/man1` in the binary distributions.
- The default type returned by `IFNULL(A,B)` [791] is now set to be the more 'general' of the types of `A` and `B`. (The order is `STRING`, `REAL` or `INTEGER`).
- Moved the `mysql.server` startup script in the RPM packages from `/etc/rc.d/init.d/mysql` to `/etc/init.d/mysql` (which almost all current Linux distributions support for LSB compliance).

- Added `Qcache_lowmem_prunes` [453] status variable (number of queries that were deleted from the cache because of low memory).
- Fixed `mysqlcheck` so it can deal with table names containing dashes.
- Bulk insert optimization (see Section 5.1.3, “Server System Variables”) is no longer used when inserting small (less than 100) number of rows.
- Optimization added for queries like `SELECT ... FROM merge_table WHERE indexed_column=constant_expr`.
- Added functions `LOCALTIME` and `LOCALTIMESTAMP` as synonyms for `NOW()` [837].
- `CEIL` is now an alias for `CEILING`.
- The `CURRENT_USER()` [872] function can be used to get a `user@host` value as it was matched in the `GRANT` system. See Section 11.13, “Information Functions”.
- Fixed `CHECK` constraints to be compatible with standard SQL. This made `CHECK` a reserved word. (Checking of `CHECK` constraints is still not implemented).
- Added `CAST(... as CHAR)`.
- Added PostgreSQL compatible `LIMIT` syntax: `SELECT ... LIMIT row_count OFFSET offset`
- `mysql_change_user()` now resets the connection to the state of a fresh connect (Ie, `ROLLBACK` any active transaction, close all temporary tables, reset all user variables etc..)
- `CHANGE MASTER TO` and `RESET SLAVE` now require that slave threads both be stopped; these commands return an error if at least one of these two threads is running.

Bugs fixed:

- Fixed number of found rows returned in `multi table updates`
- Make `--lower-case-table-names` [418] default on Mac OS X as the default file system (HFS+) is case insensitive. See Section 8.2.2, “Identifier Case Sensitivity”.
- Transactions in `autocommit = 0` [406] mode didn't rotate binary log.
- A fix for the bug in a `SELECT` with joined tables with `ORDER BY` and `LIMIT` clause when `filesort` had to be used. In that case `LIMIT` was applied to `filesort` of one of the tables, although it could not be. This fix also solved problems with `LEFT JOIN`.
- `mysql_server_init()` now makes a copy of all arguments. This fixes a problem when using the embedded server in C# program.
- Fixed buffer overrun in `libmysqlclient` library that permitted a malicious MySQL server to crash the client application. (CVE-2002-1376)
- Fixed security-related bug in `mysql_change_user()` handling. All users are strongly recommended to upgrade to version 4.0.6. (CVE-2002-1374, CVE-2002-1375)
- Fixed bug that prevented `--chroot` [385] command-line option of `mysqld` from working.
- Fixed bug in phrase operator `"..."` in boolean full-text search.
- Fixed bug that caused `OPTIMIZE TABLE` to corrupt the table under some rare circumstances.

- Part rewrite of multiple-table-update to optimize it, make it safer and more bug-free.
- `LOCK TABLES` now works together with multiple-table-update and multiple-table-delete.
- `--replicate-do=xxx` didn't work for `UPDATE` commands. (Bug introduced in 4.0.0)
- Fixed shutdown problem on Mac OS X.
- Major `InnoDB` bugs in `REPLACE, AUTO_INCREMENT, INSERT INTO ... SELECT ...` were fixed. See the `InnoDB` changelog in the `InnoDB` section of the manual.
- `RESET SLAVE` caused a crash if the slave threads were running.

C.2.27 Changes in Release 4.0.5 (13 November 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

Functionality added or changed:

- Port number was added to host name (if it is known) in `SHOW PROCESSLIST` statement.
- Changed handling of last argument in `WEEK()` [843] so that you can get week number according to the ISO 8601 specification. (Old code should still work).
- Fixed that `INSERT DELAYED` threads don't hang on `Waiting for INSERT` when one sends a `SIGHUP` to `mysqld`.
- Change that `AND` [787] works according to standard SQL when it comes to `NULL` handling. In practice, this affects only queries where you do something like `WHERE ... NOT (NULL AND 0)`.
- `mysqld` now resolves `basedir` [406] to its full path (with `realpath()`). This enables one to use relative symlinks to the MySQL installation directory. This however causes `show variables` to report different directories on systems where there is a symbolic link in the path.
- Fixed that MySQL does not use index scan on index disabled with `IGNORE INDEX` or `USE INDEX`. to be ignored.
- Added `--use-frm` [287] option to `mysqlcheck`. When used with `REPAIR TABLE`, it gets the table structure from the `.frm` file, so the table can be repaired even if the `.MYI` header is corrupted.
- Fixed bug in `MAX()` [884] optimization when used with `JOIN` and `ON` expressions.
- Added support for reading of MySQL 4.1 table definition files.
- `BETWEEN` behavior changed (see Section 11.3.2, “Comparison Functions and Operators”). Now `datetime_col BETWEEN timestamp AND timestamp` should work as expected.
- One can create `TEMPORARY MERGE` tables now.
- `DELETE FROM myisam_table` now shrinks not only the `.MYD` file but also the `.MYI` file.
- When one uses the `--open-files-limit=val` [245] option to `mysqld_safe` it is now passed on to `mysqld`.
- Changed output from `EXPLAIN` from `'where used'` to `'Using where'` to make it more in line with other output.

- Removed variable `safe_show_database` [426] as it was no longer used.
- Updated source tree to be built using `automake` 1.5 and `libtool` 1.4.
- Fixed an inadvertently changed option (`--ignore-space`) back to the original `--ignore-spaces` in `mysqlclient`. (Both syntaxes work).
- Do not require `UPDATE` [493] privilege when using `REPLACE`.
- Added support for `DROP TEMPORARY TABLE ...`, to be used to make replication safer.
- When transactions are enabled, all commands that update temporary tables inside a `BEGIN/COMMIT` are now stored in the binary log on `COMMIT` and not stored if one does `ROLLBACK`. This fixes some problems with nontransactional temporary tables used inside transactions.
- Permit braces in joins in all positions. Formerly, things like `SELECT * FROM (t2 LEFT JOIN t3 USING (a)), t1` worked, but not `SELECT * FROM t1, (t2 LEFT JOIN t3 USING (a))`. Note that braces are simply removed, they do not change the way the join is executed.
- `InnoDB` now supports also isolation levels `READ UNCOMMITTED` [975] and `READ COMMITTED` [975]. For a detailed `InnoDB` changelog, see Section C.4, “Changes in `InnoDB`”.

Bugs fixed:

- Fixed bug in `MAX()` [884] optimization when used with `JOIN` and `ON` expressions.
- Fixed that `INSERT DELAY` threads don't hang on `Waiting for INSERT` when one sends a `SIGHUP` to `mysqld`.
- Fixed that MySQL does not use an index scan on an index that has been disabled with `IGNORE INDEX` or `USE INDEX`.
- Corrected test for `root` user in `mysqld_safe`.
- Fixed error message issued when storage engine cannot do `CHECK TABLE` or `REPAIR TABLE`.
- Fixed rare core dump problem in complicated `GROUP BY` queries that didn't return any result.
- Fixed `mysqlshow` to work properly with wildcarded database names and with database names that contain underscores.
- Portability fixes to get MySQL to compile cleanly with Sun Forte 5.0.
- Fixed `MyISAM` crash when using dynamic-row tables with huge numbers of packed columns.
- Fixed query cache behavior with `BDB` transactions.
- Fixed possible floating point exception in `MATCH` relevance calculations.
- Fixed bug in full-text search `IN BOOLEAN MODE` that made `MATCH` to return incorrect relevance value in some complex joins.
- Fixed a bug that limited `MyISAM` key length to a value slightly less that 500. It is exactly 500 now.
- Fixed that `GROUP BY` on columns that may have a `NULL` value doesn't always use disk based temporary tables.
- The file name argument for the `--des-key-file` [386] argument to `mysqld` is interpreted relative to the data directory if given as a relative path name.

- Removed a condition that temp table with index on column that can be `NULL` has to be `MyISAM`. This was okay for 3.23, but not needed in 4.*. This resulted in slowdown in many queries since 4.0.2.
- Small code improvement in multiple-table updates.
- Fixed a newly introduced bug that caused `ORDER BY ... LIMIT row_count` to not return all rows.
- Fixed a bug in multiple-table deletes when outer join is used on an empty table, which gets first to be deleted.
- Fixed a bug in multiple-table updates when a single table is updated.
- Fixed bug that caused `REPAIR TABLE` and `myisamchk` to corrupt `FULLTEXT` indexes.
- Fixed bug with caching the `mysql` grant table database. Now queries in this database are not cached in the query cache.
- Small fix in `mysqld_safe` for some shells.
- Give error if a `MyISAM MERGE` table has more than 2^{32} rows and MySQL was not compiled with `-DBIG_TABLES`.
- Fixed some `ORDER BY ... DESC` problems with `InnoDB` tables.

C.2.28 Changes in Release 4.0.4 (29 September 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed bug where `GRANT`/`REVOKE` failed if host name was given in nonmatching case.
- Do not give warning in `LOAD DATA INFILE` when setting a `timestamp` [432] to a string value of `'0'`.
- Fixed bug in `myisamchk -R` mode.
- Fixed bug that caused `mysqld` to crash on `REVOKE`.
- Fixed bug in `ORDER BY` when there is a constant in the `SELECT` statement.
- One didn't get an error message if `mysqld` couldn't open the privilege tables.
- `SET PASSWORD FOR ...` closed the connection in case of errors (bug from 4.0.3).
- Increased maximum possible `max_allowed_packet` [418] in `mysqld` to 1GB.
- Fixed bug when doing a multiple-row `INSERT` on a table with an `AUTO_INCREMENT` key which was not in the first part of the key.
- Changed `LOAD DATA INFILE` to not re-create index if the table had rows from before.
- Fixed overrun bug when calling `AES_DECRYPT()` [865] with incorrect arguments.
- `--skip-ssl` [521] can now be used to disable SSL in the MySQL clients, even if one is using other SSL options in an option file or previously on the command line.
- Fixed bug in `MATCH ... AGAINST( ... IN BOOLEAN MODE)` used with `ORDER BY`.

- Added `LOCK TABLES` and `CREATE TEMPORARY TABLES` [491] privilege on the database level. You must run the `mysql_fix_privilege_tables` script on old installations to activate these.
- In `SHOW TABLE ... STATUS`, compressed tables sometimes showed up as `dynamic`.
- `SELECT @@[global|session].var_name` didn't report `global | session` in the result column name.
- Fixed problem in replication that `FLUSH LOGS` in a circular replication setup created an infinite number of binary log files. Now a `rotate-binary-log` command in the binary log does not cause slaves to rotate logs.
- Removed `STOP EVENT` from binary log when doing `FLUSH LOGS`.
- Disabled the use of `SHOW NEW MASTER FOR SLAVE` as this needs to be completely reworked in a future release.
- Fixed a bug with constant expression (for example, column of a one-row table, or column from a table, referenced by a `UNIQUE` key) appeared in `ORDER BY` part of `SELECT DISTINCT`.
- `--log-bin=a.b.c` [1181] now properly strips off `.b.c`.
- `FLUSH LOGS` removed numeric extension for all future update logs.
- `GRANT ... REQUIRE` didn't store the SSL information in the `mysql.user` table if SSL was not enabled in the server.
- `GRANT ... REQUIRE NONE` can now be used to remove SSL information.
- `AND` is now optional between `REQUIRE` options.
- `REQUIRE` option was not properly saved, which could cause strange output in `SHOW GRANTS`.
- Fixed that `mysqld --help` reports correct values for `--datadir` [385] and `--bind-address` [384].
- Fixed that one can drop UDFs that didn't exist when `mysqld` was started.
- Fixed core dump problem with `SHOW VARIABLES` on some 64-bit systems (like Solaris SPARC).
- Fixed a bug in `my_getopt()`; `--set-variable` syntax didn't work for those options that didn't have a valid variable in the `my_option` struct. This affected at least the `default-table-type` option.
- Fixed a bug from 4.0.2 that caused `REPAIR TABLE` and `myisamchk --recover` to fail on tables with duplicates in a unique key.
- Fixed a bug from 4.0.3 in calculating the default data type for some functions. This affected queries of type `CREATE TABLE tbl_name SELECT expression(),...`
- Fixed bug in queries of type `SELECT * FROM table-list GROUP BY ...` and `SELECT DISTINCT * FROM ...`.
- Fixed bug with the `--log-slow-queries` [388] option when logging an administrator command (like `FLUSH TABLES`).
- Fixed a bug that `OPTIMIZE TABLE` of locked and modified table, reported table corruption.
- Fixed a bug in `my_getopt()` in handling of special prefixes (`--skip-`, `--enable-`). `--skip-external-locking` [392] didn't work and the bug may have affected other similar options.
- Fixed bug in checking for output file name of the `tee` option.

- Added some more optimization to use index for `SELECT ... FROM many_tables .. ORDER BY key limit #`
- Fixed problem in `SHOW OPEN TABLES` when a user didn't have access permissions to one of the opened tables.

C.2.29 Changes in Release 4.0.3 (26 August 2002: Beta)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem with types of user variables. (Bug #551)
- Fixed problem with `configure ... --localstatedir=...`.
- Cleaned up `mysql.server` script.
- Fixed a bug in `mysqladmin shutdown` when pid file was modified while `mysqladmin` was still waiting for the previous one to disappear. This could happen during a very quick restart and caused `mysqladmin` to hang until `shutdown_timeout` seconds had passed.
- Do not increment warnings when setting `AUTO_INCREMENT` columns to `NULL` in `LOAD DATA INFILE`.
- Fixed all boolean type variables/options to work with the old syntax, for example, all of these work: `--lower-case-table-names` [418], `--lower-case-table-names=1` [418], `-O lower-case-table-names=1`, `--set-variable=lower-case-table-names=1`
- Fixed shutdown problem (SIGTERM signal handling) on Solaris. (Bug from 4.0.2).
- `SHOW MASTER STATUS` now returns an empty set if binary log is not enabled.
- `SHOW SLAVE STATUS` now returns an empty set if slave is not initialized.
- Do not update `MyISAM` index file on update if not strictly necessary.
- Fixed bug in `SELECT DISTINCT ... FROM many_tables ORDER BY not-used-column`.
- Fixed a bug with `BIGINT` values and quoted strings.
- Added `QUOTE()` [800] function that performs SQL quoting to produce values that can be used as data values in queries.
- Changed variable `DELAY_KEY_WRITE` to an enumeration to allow it to be set for all tables without taking down the server.
- Changed behavior of `IF(condition,column,NULL)` so that it returns the value in the column's data type.
- Made `safe_mysqld` a symlink to `mysqld_safe` in binary distribution.
- Fixed security bug when having an empty database name in the `user.db` table.
- Fixed some problems with `CREATE TABLE ... SELECT function()`.
- `mysqld` now has the option `--temp-pool` [394] enabled by default as this gives better performance with some operating systems.

- Fixed problem with too many allocated alarms on slave when connecting to master many times (normally not a very critical error).
- Fixed hang in `CHANGE MASTER TO` if the slave thread died very quickly.
- Big cleanup in replication code (less logging, better error messages, etc..)
- If the `--code-file` option is specified, the server calls `setrlimit()` to set the maximum permitted core file size to unlimited, so core files can be generated.
- Fixed bug in query cache after temporary table creation.
- Added `--count=N` [280] (`-c`) option to `mysqladmin`, to make the program do only *`N`* iterations. To be used with `--sleep` [281] (`-i`). Useful in scripts.
- Fixed bug in multiple-table `UPDATE`: when updating a table, `do_select()` became confused about reading records from a cache.
- Fixed bug in multiple-table `UPDATE` when several columns were referenced from a single table
- Fixed bug in truncating nonexistent table.
- Fixed bug in `REVOKE` that caused user resources to be randomly set.
- Fixed bug in `GRANT` for the new `CREATE TEMPORARY TABLES` [491] privilege.
- Fixed bug in multiple-table `DELETE` when tables are re-ordered in the table initialization method and ref_lengths are of different sizes.
- Fixed two bugs in `SELECT DISTINCT` with large tables.
- Fixed bug in query cache initialization with very small query cache size.
- Permit `DEFAULT` with `INSERT` statement.
- The startup parameters `myisam_max_sort_file_size` [421] and `myisam_max_extra_sort_file_size` [421] are now given in bytes, not megabytes.
- External system locking of `MyISAM`/`ISAM` files is now turned off by default. One can turn this on with `--external-locking` [387]. (For most users this is never needed).
- Fixed core dump bug with `INSERT ... SET db_name.tbl_name.col_name=''`.
- Fixed client hangup bug when using some SQL statements with incorrect syntax.
- Fixed a timing bug in `DROP DATABASE`
- New `SET [GLOBAL | SESSION]` syntax to change thread-specific and global system variables at runtime.
- Added variable `slave_compressed_protocol` [1180].
- Renamed variable `query_cache_startup_type` to `query_cache_type` [424], `myisam_bulk_insert_tree_size` to `bulk_insert_buffer_size` [408], `record_buffer` to `read_buffer_size` [425] and `record_rnd_buffer` to `read_rnd_buffer_size` [426].
- Renamed some SQL variables, but old names still work until 5.0. See Section 2.11.1.2, “Upgrading from MySQL 3.23 to 4.0”.

- Renamed `--skip-locking` to `--skip-external-locking` [392].
- Removed unused variable `query_buffer_size`.
- Fixed a bug that made the pager option in the `mysql` client nonfunctional.
- Added full `AUTO_INCREMENT` support to `MERGE` tables.
- Extended `LOG()` [820] function to accept an optional arbitrary base parameter. See Section 11.6.2, “Mathematical Functions”.
- Added `LOG2()` [821] function (useful for finding out how many bits a number would require for storage).
- Added `LN()` [820] natural logarithm function for compatibility with other databases. It is synonymous with `LOG(X)` [820].

C.2.30 Changes in Release 4.0.2 (01 July 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Cleaned up `NULL` handling for default values in `DESCRIBE tbl_name`.
- Fixed `TRUNCATE()` [825] to round up negative values to the nearest integer.
- Fixed buffer overflow problem if someone specified a too-long `--datadir` [385] option to `mysqld`. (CVE-2002-0969)
- Changed `--chroot=path` [385] option to execute `chroot()` immediately after all options have been parsed.
- Do not allow database names that contain “`\`”.
- `lower_case_table_names` [418] now also applies to database names.
- Added `XOR` [788] operator (logical and bitwise `XOR` [788]) with `^` [863] as a synonym for bitwise `XOR` [788].
- Added function `IS_FREE_LOCK('lock_name')` [879]. Based on code contributed by Hartmut Holzgraefe `<hartmut@six.de>`.
- Removed `mysql_ssl_clear()` from C API, as it was not needed.
- `DECIMAL` and `NUMERIC` types can now read exponential numbers.
- Added `SHA1()` [869] function to calculate 160 bit hash value as described in RFC 3174 (Secure Hash Algorithm). This function can be considered a cryptographically more secure equivalent of `MD5()` [868]. See Section 11.12, “Encryption and Compression Functions”.
- Added `AES_ENCRYPT()` [865] and `AES_DECRYPT()` [865] functions to perform encryption according to AES standard (Rijndael). See Section 11.12, “Encryption and Compression Functions”.
- Added `--single-transaction` [298] option to `mysqldump`, allowing a consistent dump of `InnoDB` tables. See Section 4.5.4, “`mysqldump` — A Database Backup Program”.
- Fixed bug in `innodb_log_group_home_dir` [1071] in `SHOW VARIABLES`.

- Fixed a bug in optimizer with merge tables when nonunique values are used in summing up (causing crashes).
- Fixed a bug in optimizer when a range specified makes index grouping impossible (causing crashes).
- Fixed a rare bug when `FULLTEXT` index is present and no tables are used.
- Added privileges `CREATE TEMPORARY TABLES` [491], `EXECUTE` [491], `LOCK TABLES` [492], `REPLICATION CLIENT` [492], `REPLICATION SLAVE` [492], `SHOW DATABASES` [492] and `SUPER` [493]. To use these, you must run the `mysql_fix_privilege_tables` script after upgrading.
- Fixed query cache align data bug.
- Fixed mutex bug in replication when reading from master fails.
- Added missing mutex in `TRUNCATE TABLE`. This fixes some core dump/hangup problems when using `TRUNCATE TABLE`.
- Fixed bug in multiple-table `DELETE` when optimizer uses only indexes.
- Fixed that `ALTER TABLE tbl_name RENAME new_tbl_name` is as fast as `RENAME TABLE`.
- Fixed bug in `GROUP BY` with two or more columns, where at least one column can contain `NULL` values.
- Use Turbo Boyer-Moore algorithm to speed up `LIKE "%keyword%"` [804] searches.
- Fixed bug in `DROP DATABASE` with symlink.
- Fixed crash in `REPAIR ... USE_FRM`.
- Fixed bug in `EXPLAIN` with `LIMIT offset != 0`.
- Fixed bug in phrase operator `"..."` in boolean full-text search.
- Fixed bug that caused duplicated rows when using truncation operator `*` in boolean full-text search.
- Fixed bug in truncation operator of boolean full-text search (incorrect results when there are only `+word*`s in the query).
- Fixed bug in boolean full-text search that caused a crash when an identical `MATCH` expression that did not use an index appeared twice.
- Query cache is now automatically disabled in `mysqldump`.
- Fixed problem on Windows 98 that made sending of results very slow.
- Boolean full-text search weighting scheme changed to something more reasonable.
- Fixed bug in boolean full-text search that caused MySQL to ignore queries of `ft_min_word_len` [412] characters.
- Boolean full-text search now supports “phrase searches.”
- New configure option `--without-query-cache`.
- Memory allocation strategy for “root memory” changed. Block size now grows with number of allocated blocks.
- `INET_NTOA()` [878] now returns `NULL` if you give it an argument that is too large (greater than the value corresponding to `255.255.255.255`).

- Fix `SQL_CALC_FOUND_ROWS` to work with `UNION`. It works only if the first `SELECT` has this option and if there is global `LIMIT` for the entire statement. For the moment, this requires using parentheses for individual `SELECT` queries within the statement.
- Fixed bug in `SQL_CALC_FOUND_ROWS` and `LIMIT`.
- Do not give an error for `CREATE TABLE ...(... VARCHAR(0))`.
- Fixed `SIGINT` and `SIGQUIT` problems in `mysql.cc` on Linux with some `glibc` versions.
- Fixed bug in `convert.cc`, which is caused by having an incorrect `net_store_length()` linked in the `CONVERT::store()` method.
- `DOUBLE` and `FLOAT` columns now honor the `UNSIGNED` flag on storage.
- `InnoDB` now retains foreign key constraints through `ALTER TABLE` and `CREATE/DROP INDEX`.
- `InnoDB` now enables foreign key constraints to be added through the `ALTER TABLE` syntax.
- `InnoDB` tables can now be set to automatically grow in size (auto-extend).
- Added `--ignore-lines=`*n* [304] option to `mysqlimport`. This has the same effect as the `IGNORE` *n* `LINES` clause for `LOAD DATA`.
- Fixed bug in `UNION` with last offset being transposed to total result set.
- `REPAIR ... USE_FRM` added.
- Fixed that `DEFAULT_SELECT_LIMIT` is always imposed on `UNION` result set.
- Fixed that some `SELECT` options can appear only in the first `SELECT`.
- Fixed bug with `LIMIT` with `UNION`, where last select is in the braces.
- Fixed that full-text works fine with `UNION` operations.
- Fixed bug with indexless boolean full-text search.
- Fixed bug that sometimes appeared when full-text search was used with `const` [567] tables.
- Fixed incorrect error value when doing a `SELECT` with an empty `HEAP` table.
- Use `ORDER BY column DESC` now sorts `NULL` values first. (In other words, `NULL` values sort first in all cases, whether or not `DESC` is specified.) This is changed back in 4.0.10.
- Fixed bug in `WHERE key_name='constant' ORDER BY key_name DESC`.
- Fixed bug in `SELECT DISTINCT ... ORDER BY DESC` optimization.
- Fixed bug in `... HAVING 'GROUP_FUNCTION'(`*xxx*`) IS [NOT] NULL`.
- Fixed bug in truncation operator for boolean full-text search.
- Permit value of `--user` [395] option for `mysqld` to be specified as a numeric user ID (`--user=`*user_id* [395]).
- Fixed a bug where `SQL_CALC_ROWS` returned an incorrect value when used with one table and `ORDER BY` and with `InnoDB` tables.
- Fixed that `SELECT 0 LIMIT 0` doesn't hang thread.

- Fixed some problems with `USE/IGNORE INDEX` when using many keys with the same start column.
- Do not use table scan with `BerkeleyDB` and `InnoDB` tables when we can use an index that covers the whole row.
- Optimized `InnoDB` sort-buffer handling to take less memory.
- Fixed bug in multiple-table `DELETE` and `InnoDB` tables.
- Fixed problem with `TRUNCATE TABLE` and `InnoDB` tables that produced the error `Can't execute the given command because you have active locked tables or an active transaction`.
- Added `NO_UNSIGNED_SUBTRACTION` [459] to the set of flags that may be specified with the `--sql-mode` [394] option for `mysqld`. It disables unsigned arithmetic rules when it comes to subtraction. (This makes MySQL 4.0 behave more like 3.23 with `UNSIGNED` columns).
- The result returned for all bit operators (`|` [862], `<<` [863], ...) is now of type `unsigned integer`.
- Added detection of `nan` values in `MyISAM` to make it possible to repair tables with `nan` in float or double columns.
- Fixed new bug in `myisamchk` where it didn't correctly update number of “parts” in the `MyISAM` index file.
- Changed to use `autoconf` 2.52 (from `autoconf` 2.13).
- Fixed optimization problem where the MySQL Server was in “preparing” state for a long time when selecting from an empty table which had contained a lot of rows.
- Fixed bug in complicated join with `const` [567] tables. This fix also improves performance a bit when referring to another table from a `const` [567] table.
- First pre-version of multiple-table `UPDATE` statement.
- Fixed bug in multiple-table `DELETE`.
- Fixed bug in `SELECT CONCAT(argument_list) ... GROUP BY 1`.
- `INSERT ... SELECT` did a full rollback in case of an error. Fixed so that we only roll back the last statement in the current transaction.
- Fixed bug with empty expression for boolean full-text search.
- Fixed core dump bug in updating full-text key from/to `NULL`.
- ODBC compatibility: Added `BIT_LENGTH()` [794] function.
- Fixed core dump bug in `GROUP BY BINARY column`.
- Added support for `NULL` keys in `HEAP` tables.
- Use index for `ORDER BY` in queries of type: `SELECT * FROM t WHERE key_part1=1 ORDER BY key_part1 DESC, key_part2 DESC`
- Fixed bug in `FLUSH QUERY CACHE`.
- Added `CAST()` [859] and `CONVERT()` [859] functions. The `CAST` and `CONVERT` functions are nearly identical and mainly useful when you want to create a column with a specific type in a `CREATE ... SELECT` statement. For more information, read Section 11.10, “Cast Functions and Operators”.

- `CREATE ... SELECT` on `DATE` and `TIME` functions now create columns of the expected type.
- Changed order in which keys are created in tables.
- Added new columns `Null` and `Index_type` to `SHOW INDEX` output.
- Added `--no-beep` [261] and `--prompt` [263] options to `mysql` command-line client.
- New feature: management of user resources.

```
GRANT ... WITH MAX_QUERIES_PER_HOUR N1
               MAX_UPDATES_PER_HOUR N2
               MAX_CONNECTIONS_PER_HOUR N3;
```

  See Section 5.6.4, “Setting Account Resource Limits”.

- Added `mysql_secure_installation` to the `scripts/` directory.

C.2.31 Changes in Release 4.0.1 (23 December 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `system` [566] command to `mysql`.
- Fixed bug when `HANDLER` was used with some unsupported table type.
- `mysqldump` now puts `ALTER TABLE tbl_name DISABLE KEYS` and `ALTER TABLE tbl_name ENABLE KEYS` in the sql dump.
- Added `mysql_fix_extensions` script.
- Fixed stack overrun problem with `LOAD DATA FROM MASTER` on OSF/1.
- Fixed shutdown problem on HP-UX.
- Added `DES_ENCRYPT()` [866] and `DES_DECRYPT()` [866] functions.
- Added `FLUSH DES_KEY_FILE` statement.
- Added `--des-key-file` [386] option to `mysqld`.
- `HEX(str)` [796] now returns the characters in *`str`* converted to hexadecimal.
- Fixed problem with `GRANT` when using `lower_case_table_names = 1`.
- Changed `SELECT ... IN SHARE MODE` to `SELECT ... LOCK IN SHARE MODE` (as in MySQL 3.23).
- A new query cache to cache results from identical `SELECT` queries.
- Fixed core dump bug on 64-bit machines when it got an incorrect communication packet.
- `MATCH ... AGAINST(... IN BOOLEAN MODE)` can now work without `FULLTEXT` index.
- Fixed slave to replicate from 3.23 master.

- Miscellaneous replication fixes/cleanup.
- Got shutdown to work on Mac OS X.
- Added `myisam/ft_dump` utility for low-level inspection of `FULLTEXT` indexes.
- Fixed bug in `DELETE ... WHERE ... MATCH ...`.
- Added support for `MATCH ... AGAINST(... IN BOOLEAN MODE)`. Note that you must rebuild your tables with `ALTER TABLE tbl_name TYPE=MyISAM` to be able to use boolean full-text search.
- `LOCATE()` [798] and `INSTR()` [797] are now case sensitive if either argument is a binary string.
- Changed `RAND()` [822] initialization so that `RAND(N)` [822] and `RAND(N+1)` [822] are more distinct.
- Fixed core dump bug in `UPDATE ... ORDER BY`.
- In 3.23, `INSERT INTO ... SELECT` always had `IGNORE` enabled. Now MySQL stops (and possibly rolls back) by default in case of an error unless you specify `IGNORE`.
- Ignore `DATA DIRECTORY` and `INDEX DIRECTORY` directives on Windows.
- Added boolean full-text search code. It should be considered early alpha.
- Extended `MODIFY` and `CHANGE` in `ALTER TABLE` to accept the `FIRST` and `AFTER` keywords.
- Indexes are now used with `ORDER BY` on a whole `InnoDB` table.

C.2.32 Changes in Release 4.0.0 (October 2001: Alpha)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `--xml` [265] option to `mysql` for producing XML output.
- Added full-text variables `ft_min_word_len` [412], `ft_max_word_len` [412], and `ft_max_word_len_for_sort` system variables.
- Added full-text variables `ft_min_word_len` [412], `ft_max_word_len` [412], and `ft_max_word_len_for_sort` variables to `myisamchk`.
- Added documentation for `libmysqld`, the embedded MySQL server library. Also added example programs (a `mysql` client and `mysqltest` test program) which use `libmysqld`.
- Removed all Gemini hooks from MySQL server.
- Removed `my_thread_init()` and `my_thread_end()` from `mysql_com.h`, and added `mysql_thread_init()` and `mysql_thread_end()` to `mysql.h`.
- Support for communication packets > 16MB. In 4.0.1 we extended `MyISAM` to be able to handle these.
- Secure connections (with SSL).
- Unsigned `BIGINT` constants now work. `MIN()` [884] and `MAX()` [884] now handle signed and unsigned `BIGINT` numbers correctly.
- New character set `latin1_de` which provides correct German sorting.

- `STRCMP()` [807] now uses the current character set when doing comparisons, which means that the default comparison behavior now is case insensitive.
- `TRUNCATE TABLE` and `DELETE FROM tbl_name` are now separate functions. One bonus is that `DELETE FROM tbl_name` now returns the number of deleted rows, rather than zero.
- `DROP DATABASE` now executes a `DROP TABLE` on all tables in the database, which fixes a problem with `InnoDB` tables.
- Added support for `UNION`.
- Added support for multiple-table `DELETE` operations.
- A new `HANDLER` interface to `MyISAM` tables.
- Added support for `INSERT` on `MERGE` tables. Patch from Benjamin Pflugmann.
- Changed `WEEK(date,0)` [843] to match the calendar in the USA.
- `COUNT(DISTINCT)` [882] is about 30% faster.
- Speed up all internal list handling.
- Speed up `IS NULL` [783], `ISNULL()` [785] and some other internal primitives.
- Full-text index creation now is much faster.
- Tree-like cache to speed up bulk inserts and `myisam_bulk_insert_tree_size` variable.
- Searching on packed (`CHAR`/`VARCHAR`) keys is now much faster.
- Optimized queries of type: `SELECT DISTINCT * FROM tbl_name ORDER by key_part1 LIMIT row_count`.
- `SHOW CREATE TABLE` now shows all table attributes.
- `ORDER BY ... DESC` can now use keys.
- `LOAD DATA FROM MASTER` “automatically” sets up a slave.
- Renamed `safe_mysqld` to `mysqld_safe` to make this name more in line with other MySQL scripts/commands.
- Added support for symbolic links to `MyISAM` tables. Symlink handling is now enabled by default for Windows.
- Added `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` [872]. This makes it possible to know how many rows a query would have returned without a `LIMIT` clause.
- Changed output format of `SHOW OPEN TABLES`.
- Permit `SELECT expression LIMIT ...`.
- Added the `identity` [414] variable as a synonym for the `last_insert_id` [416] variable (like Sybase).
- Added `ORDER BY` syntax to `UPDATE` and `DELETE`.
- `SHOW INDEXES` is now a synonym for `SHOW INDEX`.

- Added `ALTER TABLE tbl_name DISABLE KEYS` and `ALTER TABLE tbl_name ENABLE KEYS` commands.
- Permit use of `IN` as a synonym for `FROM` in `SHOW` commands.
- Implemented “repair by sort” for `FULLTEXT` indexes. `REPAIR TABLE`, `ALTER TABLE`, and `OPTIMIZE TABLE` for tables with `FULLTEXT` indexes are now up to 100 times faster.
- Permit standard SQL syntax `X'hexadecimal-number'`.
- Cleaned up global lock handling for `FLUSH TABLES WITH READ LOCK`.
- Fixed problem with `DATETIME = constant` in `WHERE` optimization.
- Added `--master-data` [296] and `--no-autocommit` [297] options to `mysqldump`. (Thanks to Brian Aker for this.)
- Added script `mysql_explain_log.sh` to distribution. (Thanks to mobile.de).

C.3 Changes in Release 3.23.x (Lifecycle Support Ended)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

The 3.23 release has several major features that are not present in previous versions. We have added three new table types:

- `MyISAM`

  A new `ISAM` library which is tuned for SQL and supports large files.

- `InnoDB`

  A transaction-safe storage engine that supports row level locking, and many Oracle-like features.

- `BerkeleyDB` or `BDB`

  Uses the Berkeley DB library from Sleepycat Software to implement transaction-safe tables.

Note that only `MyISAM` is available in the standard binary distribution.

The 3.23 release also includes support for database replication between a master and many slaves, full-text indexing, and much more.

C.3.1 Changes in Release 3.23.59 (Not released)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed an old bug in concurrent accesses to `MERGE` tables (even one `MERGE` table and `MyISAM` tables), that could've resulted in a crash or hang of the server. (Bug #2408, CVE-2004-0837)
- Fixed incorrect destruction of expression which led to crash of server on complex `AND` [787]/`OR` [787] expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969, Bug #4494)
- Fixed problem with parsing complex queries on 64-bit architectures. (Bug #4204)
- Fixed a symlink vulnerability in the `mysqlbug` script. (Bug #3284, CVE-2004-0381)
- Fixed bug in privilege checking of `ALTER TABLE RENAME`. (Bug #3270, CVE-2004-0835)
- Fixed bugs in `ACOS()` [817], `ASIN()` [818] (Bug #2338) and in `FLOOR()` [820] (Bug #3051). The cause of the problem is an overly strong optimization done by `gcc` in this case.
- Fixed bug in `INSERT ... SELECT` statements where, if a `NOT NULL` column is assigned a value of `NULL`, the following columns in the row might be assigned a value of zero. (Bug #2012)
- If a query was ignored on the slave (because of `--replicate-ignore-table` [1177] and other similar rules), the slave still checked if the query got the same error code (0, no error) as on the master. So if the master had an error on the query (for example, “Duplicate entry” in a multiple-row insert), then the slave stopped and warned that the error codes didn't match. This is a backport of the fix for MySQL 4.0. (Bug #797)
- `mysqlbinlog` now asks for a password at console when the `-p`/`--password` [341] option is used with no argument. This is how the other clients (`mysqladmin`, `mysqldump`..) behave. Note that one now has to use `mysqlbinlog -p<my_password>`; `mysqlbinlog -p <my_password>` does not work anymore (in other words, put no space after `-p`). (Bug #1595)
- On some 64-bit machines (some HP-UX and Solaris machines), a slave installed with the 64-bit MySQL binary could not connect to its master (it connected to itself instead). (Bug #1256, Bug #1381)
- Fixed a Windows-specific bug present since MySQL 3.23.57 and 3.23.58 that caused Windows slaves to crash when they started replication if a `master.info` file existed. (Bug #1720)
- Fixed bug in `ALTER TABLE RENAME`, when rename to the table with the same name in another database silently dropped destination table if it existed. (Bug #2628)
- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017, CVE-2004-0836)

C.3.2 Changes in Release 3.23.58 (11 September 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed buffer overflow in password handling which could potentially be exploited by MySQL users with `ALTER` [491] privilege on the `mysql.user` table to execute random code or to gain shell access with the UID of the `mysqld` process (thanks to Jedi/Sector One for spotting and reporting this bug). (CVE-2003-0780)
- `mysqldump` now correctly quotes all identifiers when communicating with the server. This assures that during the dump process, `mysqldump` never sends queries to the server that result in a syntax error. This problem is **not** related to the `mysqldump` program's output, which was not changed. (Bug #1148)

- Fixed table/column grant handling: The proper sort order (from most specific to less specific, see Section 5.5.5, “Access Control, Stage 2: Request Verification”) was not honored. (Bug #928)
- Fixed overflow bug in `MyISAM` and `ISAM` when a row is updated in a table with a large number of columns and at least one `BLOB/TEXT` column.
- Fixed MySQL so that field length (in C API) for the second column in `SHOW CREATE TABLE` is always larger than the data length. The only known application that was affected by the old behavior was Borland dbExpress, which truncated the output from the command. (Bug #1064)
- Fixed `ISAM` bug in `MAX()` [884] optimization.
- Fixed `Unknown error` when doing `ORDER BY` on reference table which was used with `NULL` value on `NOT NULL` column. (Bug #479)

C.3.3 Changes in Release 3.23.57 (06 June 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem in alarm handling that could cause problems when getting a packet that is too large.
- Fixed problem when installing MySQL as a service on Windows when two arguments were specified to `mysqld` (option file group name and service name).
- Fixed `kill pid-of-mysqld` to work on Mac OS X.
- `SHOW TABLE STATUS` displayed incorrect `Row_format` value for tables that have been compressed with `myisampack`. (Bug #427)
- `SHOW VARIABLES LIKE 'innodb_data_file_path'` displayed only the name of the first data file. (Bug #468)
- Fixed security problem where `mysqld` didn't allow one to `UPDATE` rows in a table even if one had a global `UPDATE` [493] privilege and a database `SELECT` [492] privilege.
- Fixed a security problem with `SELECT` and wildcarded select list, when user only had partial column `SELECT` [492] privileges on the table.
- Fixed unlikely problem in optimizing `WHERE` clause with a constant expression such as in `WHERE 1 AND (a=1 AND b=1)`.
- Fixed problem on IA-64 with timestamps that caused `mysqlbinlog` to fail.
- The default option for `innodb_flush_log_at_trx_commit` [1068] was changed from 0 to 1 to make `InnoDB` tables ACID by default. See Section 13.2.4, “`InnoDB` Startup Options and System Variables”.
- Fixed problem with too many allocated alarms on slave when connecting to master many times (normally not a very critical error).
- Fixed a bug in replication of temporary tables. (Bug #183)
- Fixed 64-bit bug that affected at least AMD hammer systems.
- Fixed a bug when doing `LOAD DATA INFILE IGNORE`: When reading the binary log, `mysqlbinlog` and the replication code read `REPLACE` instead of `IGNORE`. This could make the slave's table become different from the master's table. (Bug #218)

- Fixed overflow bug in MyISAM when a row is inserted into a table with a large number of columns and at least one BLOB/TEXT column. Bug was caused by incorrect calculation of the needed buffer to pack data.
- The binary log was not locked during TRUNCATE *tbl_name* or DELETE FROM *tbl_name* statements, which could cause an INSERT to *tbl_name* to be written to the log before the TRUNCATE TABLE or DELETE statements.
- Fixed rare bug in UPDATE of InnoDB tables where one row could be updated multiple times.
- Produce an error for empty table and column names.
- Changed PROCEDURE ANALYSE() to report DATE instead of NEWDATE.
- Changed PROCEDURE ANALYSE(#) to restrict the number of values in an ENUM column to # also for string values.
- mysqldump no longer silently deletes the binary logs when invoked with the --master-data [296] or --first-slave [294] option; while this behavior was convenient for some users, others may suffer from it. Now you must explicitly ask for binary logs to be deleted by using the new --delete-master-logs [294] option.
- Fixed a bug in mysqldump when it was invoked with the --master-data [296] option: The CHANGE MASTER TO statements that were appended to the SQL dump had incorrect coordinates. (Bug #159)

C.3.4 Changes in Release 3.23.56 (13 March 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed mysqld crash on extremely small values of sort_buffer variable.
- Fixed a bug in privilege system for GRANT UPDATE on the column level.
- Fixed a rare bug when using a date in HAVING with GROUP BY.
- Fixed checking of random part of WHERE clause. (Bug #142)
- Fixed MySQL (and myisamchk) crash on artificially corrupted .MYI files.
- Security enhancement: mysqld no longer reads options from world-writable config files. (CVE-2003-0150)
- Security enhancement: mysqld and safe_mysqld now use only the first --user option specified on the command line. Normally this comes from /etc/my.cnf. (CVE-2003-0150)
- Security enhancement: Do not allow BACKUP TABLE to overwrite existing files.
- Fixed unlikely deadlock bug when one thread did a LOCK TABLE and another thread did a DROP TABLE. In this case one could do a KILL on one of the threads to resolve the deadlock.
- LOAD DATA INFILE was not replicated by slave if replicate_*_table was set on the slave.
- Fixed a bug in handling CHAR(0) columns that could cause incorrect results from the query.
- Fixed a bug in SHOW VARIABLES on 64-bit platforms. The bug was caused by incorrect declaration of variable server_id [426].

- The `Comment` column in `SHOW TABLE STATUS` now reports that it can contain `NULL` values (which is the case for a crashed `.frm` file).
- Fixed the `rpl_rotate_logs` test to not fail on certain platforms (such as Mac OS X) due to a too-long file name (changed `slave-master-info.opt` to `.slave-mi`).
- Fixed a problem with `BLOB NOT NULL` columns used with `IS NULL` [783].
- Fixed bug in `MAX()` [884] optimization in `MERGE` tables.
- Better `RAND()` [822] initialization for new connections.
- Fixed bug with connect timeout. This bug was manifested on OS's with `poll()` system call, which resulted in timeout the value specified as it was executed in both `select()` and `poll()`.
- Fixed bug in `SELECT * FROM table WHERE datetime1 IS NULL OR datetime2 IS NULL`.
- Fixed bug in using aggregate functions as argument for `INTERVAL()` [785], `CASE` [790], `FIELD()` [796], `CONCAT_WS()` [795], `ELT()` [795] and `MAKE_SET()` [799] functions.
- When running with `--lower-case-table-names=1` [418] (default on Windows) and you had tables or databases with mixed case on disk, then executing `SHOW TABLE STATUS` followed with `DROP DATABASE` or `DROP TABLE` could fail with `Errcode 13`.
- Fixed bug in logging to binary log (which affects replication) a query that inserts a `NULL` in an `auto_increment` field and also uses `LAST_INSERT_ID()` [874].
- Fixed bug in `mysqladmin --relative`.
- On some 64-bit systems, `show status` reported a strange number for `Open_files` and `Open_streams` [453].

C.3.5 Changes in Release 3.23.55 (23 January 2003)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed double `free`'d pointer bug in `mysql_change_user()` handling, that enabled a specially hacked version of MySQL client to crash `mysqld`. Note that you must log in to the server by using a valid user account to be able to exploit this bug. (CVE-2003-0073)
- Fixed bug with the `--log-slow-queries` [388] when logging an administrator command (like `FLUSH TABLES`).
- Fixed bug in `GROUP BY` when used on BLOB column with `NULL` values.
- Fixed a bug in handling `NULL` values in `CASE ... WHEN ...`.
- Bugfix for `--chroot` [385] (see Section C.3.6, “Changes in Release 3.23.54 (05 December 2002)”) is reverted. Unfortunately, there is no way to make it to work, without introducing backward-incompatible changes in `my.cnf`. Those who need `--chroot` [385] functionality, should upgrade to MySQL 4.0. (The fix in the 4.0 branch did not break backward-compatibility).
- Make `--lower-case-table-names` [418] default on Mac OS X as the default file system (HFS+) is case insensitive.

- Fixed a bug in `scripts/mysqld_safe.sh` in `NOHUP_NICENESS` testing.
- Transactions in `autocommit = 0` [406] mode didn't rotate the binary log.
- Fixed a bug in `scripts/make_binary_distribution` that resulted in a remaining `@HOSTNAME@` variable instead of replacing it with the correct path to the `hostname` binary.
- Fixed a very unlikely bug that could cause `SHOW PROCESSLIST` to core dump in pthread_mutex_unlock() if a new thread was connecting.
- Forbid `SLAVE STOP` if the thread executing the query has locked tables. This removes a possible deadlock situation.

C.3.6 Changes in Release 3.23.54 (05 December 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a bug, that enabled a specially crafted packet to crash `mysqld`. (CVE-2002-1373)
- Fixed a rare crash (double `free`'d pointer) when altering a temporary table.
- Fixed buffer overrun in `libmysqlclient` library that permitted a malicious MySQL server to crash the client application. (CVE-2002-1376)
- Fixed security-related bug in `mysql_change_user()` handling. All users are strongly recommended to upgrade to the version 3.23.54. (CVE-2002-1374, CVE-2002-1375)
- Fixed bug that prevented `--chroot` [385] command-line option of `mysqld` from working.
- Fixed bug that made `OPTIMIZE TABLE` to corrupt the table under some rare circumstances.
- Fixed `mysqlcheck` so it can deal with table names containing dashes.
- Fixed shutdown problem on Mac OS X.
- Fixed bug with comparing an indexed `NULL` field with `<=> NULL`.
- Fixed bug that caused `IGNORE INDEX` and `USE INDEX` sometimes to be ignored.
- Fixed rare core dump problem in complicated `GROUP BY` queries that didn't return any result.
- Fixed a bug where `MATCH ... AGAINST () >=0` was treated as if it was `>`.
- Fixed core dump in `SHOW PROCESSLIST` when running with an active slave (unlikely timing bug).
- Make it possible to use multiple MySQL servers on Windows (code backported from 4.0.2).
- One can create `TEMPORARY MERGE` tables now.
- Fixed that `--core-file` [385] works on Linux (at least on kernel 2.4.18).
- Fixed a problem with `BDB` and `ALTER TABLE`.
- Fixed reference to freed memory when doing complicated `GROUP BY ... ORDER BY` queries. Symptom was that `mysqld` died in function `send_fields`.

- Allocate heap rows in smaller blocks to get better memory usage.
- Fixed memory allocation bug when storing `BLOB` values in internal temporary tables used for some (unlikely) `GROUP BY` queries.
- Fixed a bug in key optimizing handling where the expression `WHERE col_name = key_col_name` was calculated as true for `NULL` values.
- Fixed core dump bug when doing `LEFT JOIN ... WHERE key_column=NULL`.
- Fixed `MyISAM` crash when using dynamic-row tables with huge numbers of packed fields.
- Updated source tree to be built using `automake` 1.5 and `libtool` 1.4.

C.3.7 Changes in Release 3.23.53 (09 October 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed crash when `SHOW INNODB STATUS` was used and `skip-innodb` was defined.
- Fixed possible memory corruption bug in binary log file handling when slave rotated the logs (only affected 3.23, not 4.0).
- Fixed problem in `LOCK TABLES` on Windows when one connects to a database that contains uppercase letters.
- Fixed that `--skip-show-database` [394] doesn't reset the `--port` [391] option.
- Small fix in `safe_mysqld` for some shells.
- Fixed that `FLUSH STATUS` doesn't reset `delayed_insert_threads`.
- Fixed core dump bug when using the `BINARY` cast on a `NULL` value.
- Fixed race condition when someone did a `GRANT` at the same time a new user logged in or did a `USE database`.
- Fixed bug in `ALTER TABLE` and `RENAME TABLE` when running with `-O lower_case_table_names=1` (typically on Windows) when giving the table name in uppercase.
- Fixed that `-O lower_case_table_names=1` also converts database names to lowercase.
- Fixed unlikely core dump with `SELECT ... ORDER BY ... LIMIT`.
- Changed `AND` [787]/`OR` [787] to report that they can return NULL. This fixes a bug in `GROUP BY` on `AND` [787]/`OR` [787] expressions that return `NULL`.
- Fixed a bug that `OPTIMIZE TABLE` of locked and modified `MyISAM` table, reported table corruption.
- Fixed a `BDB`-related `ALTER TABLE` bug with dropping a column and shutting down immediately thereafter.
- Fixed problem with `configure ... --localstatedir=...`.
- Fixed problem with `UNSIGNED BIGINT` on AIX (again).

- Fixed bug in pthread_mutex_trylock() on HPUX 11.0.
- Multi-threaded stress tests for `InnoDB`.

C.3.8 Changes in Release 3.23.52 (14 August 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Wrap `BEGIN`/`COMMIT` around transaction in the binary log. This makes replication honor transactions.
- Fixed security bug when having an empty database name in the `user.db` table.
- Changed initialization of `RAND()` [822] to make it less predicatable.
- Fixed problem with `GROUP BY` on result with expression that created a `BLOB` field.
- Fixed problem with `GROUP BY` on columns that have `NULL` values. To solve this we now create an `MyISAM` temporary table when doing a `GROUP BY` on a possible `NULL` item. From MySQL 4.0.5 we can use in memory `HEAP` tables for this case.
- Fixed problem with privilege tables when downgrading from 4.0.2 to 3.23.
- Fixed thread bug in `SLAVE START`, `SLAVE STOP` and automatic repair of `MyISAM` tables that could cause table cache to be corrupted.
- Fixed possible thread related key-cache-corruption problem with `OPTIMIZE TABLE` and `REPAIR TABLE`.
- Added name of 'administrator command' logs.
- Fixed bug with creating an auto-increment value on second part of a `UNIQUE` key where first part could contain `NULL` values.
- Do not write slave-timeout reconnects to the error log.
- Fixed bug with slave net read timeouting
- Fixed a core-dump bug with `MERGE` tables and `MAX()` [884] function.
- Fixed bug in `ALTER TABLE` with `BDB` tables.
- Fixed bug when logging `LOAD DATA INFILE` to binary log with no active database.
- Fixed a bug in range optimizer (causing crashes).
- Fixed possible problem in replication when doing `DROP DATABASE` on a database with `InnoDB` tables.
- Fixed `mysql_info()` to return 0 for `Duplicates` value when using `INSERT DELAYED IGNORE`.
- Added `-DHAVE_BROKEN_REALPATH` to the Mac OS X (darwin) compile options in `configure.in` to fix a failure under high load.

C.3.9 Changes in Release 3.23.51 (31 May 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please

consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fix bug with closing tags missing slash for `mysqldump` XML output.
- Remove endspace from `ENUM` values. (This fixed a problem with `SHOW CREATE TABLE`.)
- Fixed bug in `CONCAT_WS()` [795] that cut the result.
- Changed name of server variables `Com_show_master_stat` to `Com_show_master_status` and `Com_show_slave_stat` to `Com_show_slave_status`.
- Changed handling of `gethostbyname()` to make the client library thread-safe even if `gethostbyname_r` doesn't exist.
- Fixed core-dump problem when giving a wrong password string to `GRANT`.
- Fixed bug in `DROP DATABASE` with symlinked directory.
- Fixed optimization problem with `DATETIME` and value outside `DATETIME` range.
- Removed Sleepycat's `BDB` doc files from the source tree, as they're not needed (MySQL covers `BDB` in its own documentation).
- Fixed MIT-pthreads to compile with `glibc` 2.2 (needed for `make dist`).
- Fixed the `FLOAT(X+1,X)` is not converted to `FLOAT(X+2,X)`. (This also affected `DECIMAL`, `DOUBLE` and `REAL` types)
- Fixed the result from `IF()` [790] is case in-sensitive if the second and third arguments are case sensitive.
- Fixed core dump problem on OSF/1 in `gethostbyname_r`.
- Fixed that underflowed decimal fields are not zero filled.
- If we get an overflow when inserting `'+11111'` for `DECIMAL(5,0) UNSIGNED` columns, we just drop the sign.
- Fixed optimization bug with `ISNULL(`*`expression_which_cannot_be_null`*`)` [785] and `ISNULL(`*`constant_expression`*`)` [785].
- Fixed host lookup bug in the `glibc` library that we used with the 3.23.50 Linux-x86 binaries.

C.3.10 Changes in Release 3.23.50 (21 April 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed buffer overflow problem if someone specified a too-long `--datadir` [385] option to `mysqld`. (CVE-2002-0969)
- Add missing `<row>` tags for `mysqldump` XML output.
- Fixed problem with `crash-me` and `gcc` 3.0.4.
- Fixed that `@@unknown_variable` doesn't hang server.

- Added `@@VERSION` as a synonym for `VERSION()` [876].
- `SHOW VARIABLES LIKE 'xxx'` is now case-insensitive.
- Fixed timeout for `GET_LOCK()` [877] on HP-UX with DCE threads.
- Fixed memory allocation bug in the glibc library used to build Linux binaries, which caused `mysqld` to die in `free()`.
- Fixed `SIGINT` and `SIGQUIT` problems in `mysql`.
- Fixed bug in character table converts when used with big (larger than 64KB) strings.
- `InnoDB` now retains foreign key constraints through `ALTER TABLE` and `CREATE/DROP INDEX`.
- `InnoDB` now enables foreign key constraints to be added through the `ALTER TABLE` syntax.
- `InnoDB` tables can now be set to automatically grow in size (auto-extend).
- Our Linux RPMS and binaries are now compiled with `gcc` 3.0.4, which should make them a bit faster.
- Fixed some buffer overflow problems when reading startup parameters.
- Because of problems on shutdown we have now disabled named pipes on Windows by default. One can enable named pipes by starting `mysqld` with `--enable-named-pipe` [386].
- Fixed bug when using `WHERE key_column = 'J' or key_column='j'`.
- Fixed core-dump bug when using `--log-bin` [1181] with `LOAD DATA INFILE` without an active database.
- Fixed bug in `RENAME TABLE` when used with `lower_case_table_names=1` (default on Windows).
- Fixed unlikely core-dump bug when using `DROP TABLE` on a table that was in use by a thread that also used queries on only temporary tables.
- Fixed problem with `SHOW CREATE TABLE` and `PRIMARY KEY` when using 32 indexes.
- Fixed that one can use `SET PASSWORD` for the anonymous user.
- Fixed core dump bug when reading client groups from option files using `mysql_options()`.
- Memory leak (16 bytes per every **corrupted** table) closed.
- Fixed binary builds to use `--enable-local-infile`.
- Update source to work with new version of `bison`.
- Updated shell scripts to now agree with new POSIX standard.
- Fixed bug where `DATE_FORMAT()` [832] returned empty string when used with `GROUP BY`.

C.3.11 Changes in Release 3.23.49 (14 February 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- For a `MERGE` table, `DELETE FROM merge_table` used without a `WHERE` clause no longer clears the mapping for the table by emptying the `.MRG` file. Instead, it deletes records from the mapped tables.
- Do not give warning for a statement that is only a comment; this is needed for `mysqldump --disable-keys` to work.
- Fixed unlikely caching bug when doing a join without keys. In this case, the last used field for a table always returned `NULL`.
- Added options to make `LOAD DATA LOCAL INFILE` more secure.
- MySQL binary release 3.23.48 for Linux contained a new `glibc` library, which has serious problems under high load and Red Hat 7.2. The 3.23.49 binary release doesn't have this problem.
- Fixed shutdown problem on NT.

C.3.12 Changes in Release 3.23.48 (07 February 2002)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `--xml` [300] option to `mysqldump` for producing XML output.
- Changed to use `autoconf` 2.52 (from `autoconf` 2.13)
- Fixed bug in complicated join with `const` [567] tables.
- Added internal safety checks for `InnoDB`.
- Some `InnoDB` variables were always shown in `SHOW VARIABLES` as `OFF` on high-byte-first systems (like SPARC).
- Fixed problem with one thread using an `InnoDB` table and another thread doing an `ALTER TABLE` on the same table. Before that, `mysqld` could crash with an assertion failure in `row0row.c`, line 474.
- Tuned the `InnoDB` SQL optimizer to favor index searches more often over table scans.
- Fixed a performance problem with `InnoDB` tables when several large `SELECT` queries are run concurrently on a multiprocessor Linux computer. Large CPU-bound `SELECT` queries now also generally run faster on all platforms.
- If MySQL binary logging is used, `InnoDB` now prints after crash recovery the latest MySQL binary log name and the offset `InnoDB` was able to recover to. This is useful, for example, when resynchronizing a master and a slave database in replication.
- Added better error messages to help in installation problems of `InnoDB` tables.
- It is now possible to recover MySQL temporary tables that have become orphaned inside the `InnoDB` tablespace.
- `InnoDB` now prevents a `FOREIGN KEY` declaration where the signedness is not the same in the referencing and referenced integer columns.
- Calling `SHOW CREATE TABLE` or `SHOW TABLE STATUS` could cause memory corruption and make `mysqld` crash. Especially at risk was `mysqldump`, because it frequently calls `SHOW CREATE TABLE`.

- If inserts to several tables containing an `AUTO_INCREMENT` column were wrapped inside one `LOCK TABLES`, `InnoDB` asserted in `lock0lock.c`.
- In 3.23.47 we permitted several `NULL` values in a `UNIQUE` secondary index for an `InnoDB` table. But `CHECK TABLE` was not relaxed: it reports the table as corrupt. `CHECK TABLE` no longer complains in this situation.
- `SHOW GRANTS` now shows `REFERENCES` [492] instead of `REFERENCE`.

C.3.13 Changes in Release 3.23.47 (27 December 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed bug when using the following construct: `SELECT ... WHERE key=@var_name OR key=@var_name2`
- Restrict `InnoDB` keys to 500 bytes.
- `InnoDB` now supports `NULL` in keys.
- Fixed shutdown problem on HP-UX. (Introduced in 3.23.46)
- Fixed core dump bug in replication when using `SELECT RELEASE_LOCK()`.
- Added new statement: `DO expr[,expr]...`
- Added `slave-skip-errors` option.
- Added statistics variables for all MySQL commands. (`SHOW STATUS` is now much longer.)
- Fixed default values for `InnoDB` tables.
- Fixed that `GROUP BY expr DESC` works.
- Fixed bug when using `t1 LEFT JOIN t2 ON t2.key=constant`.
- `mysql_config` now also works with binary (relocated) distributions.

C.3.14 Changes in Release 3.23.46 (29 November 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem with aliased temporary table replication.
- `InnoDB` and `BDB` tables now use index when doing an `ORDER BY` on the whole table.
- Fixed bug where one got an empty set instead of a DEADLOCK error when using `BDB` tables.
- One can now kill `ANALYZE TABLE`, `REPAIR TABLE`, and `OPTIMIZE TABLE` when the thread is waiting to get a lock on the table.
- Fixed race condition in `ANALYZE TABLE`.

- Fixed bug when joining with caching (unlikely to happen).
- Fixed race condition when using the binary log and `INSERT DELAYED` which could cause the binary log to have rows that were not yet written to `MyISAM` tables.
- Changed caching of binary log to make replication slightly faster.
- Fixed bug in replication on Mac OS X.

C.3.15 Changes in Release 3.23.45 (22 November 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- `(UPDATE|DELETE) ...WHERE MATCH` bugfix.
- Shutdown should now work on Darwin (Mac OS X).
- Fixed core dump when repairing corrupted packed `MyISAM` files.
- `--core-file` [385] now works on Solaris.
- Fix a bug which could cause `InnoDB` to complain if it cannot find free blocks from the buffer cache during recovery.
- Fixed bug in `InnoDB` insert buffer B-tree handling that could cause crashes.
- Fixed bug in `InnoDB` lock timeout handling.
- Fixed core dump bug in `ALTER TABLE` on a `TEMPORARY InnoDB` table.
- Fixed bug in `OPTIMIZE TABLE` that reset index cardinality if it was up to date.
- Fixed problem with `t1 LEFT_JOIN t2 ... WHERE t2.date_column IS NULL` when date_column was declared as `NOT NULL`.
- Fixed bug with `BDB` tables and keys on `BLOB` columns.
- Fixed bug in `MERGE` tables on OS with 32-bit file pointers.
- Fixed bug in `TIME_TO_SEC()` [841] when using negative values.

C.3.16 Changes in Release 3.23.44 (31 October 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed `Rows_examined` count in slow query log.
- Fixed bug when using a reference to an `AVG()` [881] column in `HAVING`.
- Fixed that date functions that require correct dates, like `DAYOFYEAR(column)` [834], return `NULL` for `0000-00-00` dates.

- Fixed bug in const-propagation when comparing columns of different types. (`SELECT * FROM date_col="2001-01-01" and date_col=time_col`)
- Fixed bug that caused error message `Can't write, because of unique constraint` with some `GROUP BY` queries.
- Fixed problem with `sjis` character strings used within quoted table names.
- Fixed core dump when using `CREATE ... FULLTEXT` keys with other storage engines than `MyISAM`.
- Do not use `signal()` on Windows because this appears to not be 100% reliable.
- Fixed bug when doing `WHERE col_name=NULL` on an indexed column that had `NULL` values.
- Fixed bug when doing `LEFT JOIN ... ON (col_name = constant) WHERE col_name = constant`.
- When using replications, aborted queries that contained `%` could cause a core dump.
- `TCP_NODELAY` was not used on some systems. (Speed problem.)
- Applied portability fixes for OS/2. (Patch by Yuri Dario.)

The following changes are for `InnoDB` tables:

- Add missing `InnoDB` variables to `SHOW VARIABLES`.
- Foreign key checking is now done for `InnoDB` tables.
- `DROP DATABASE` now works also for `InnoDB` tables.
- `InnoDB` now supports data files and raw disk partitions bigger than 4GB on those operating systems that have big files.
- `InnoDB` calculates better table cardinality estimates for the MySQL optimizer.
- Accent characters in the default character set `latin1` are ordered according to the MySQL ordering.

  Note: If you are using `latin1` and have inserted characters whose code is greater than 127 into an indexed `CHAR` column, you should run `CHECK TABLE` on your table when you upgrade to 3.23.44, and drop and reimport the table if `CHECK TABLE` reports an error!
- A new `my.cnf` parameter, `innodb_thread_concurrency` [1072], helps in performance tuning in heavily concurrent environments.
- A new `my.cnf` parameter, `innodb_fast_shutdown` [1068], speeds up server shutdown.
- A new `my.cnf` parameter, `innodb_force_recovery` [1069], helps to save your data in case the disk image of the database becomes corrupt.
- `innodb_monitor` has been improved and a new `innodb_table_monitor` added.
- Increased maximum key length from 500 to 7000 bytes.
- Fixed a bug in replication of `AUTO_INCREMENT` columns with multiple-line inserts.
- Fixed a bug when the case of letters changes in an update of an indexed secondary column.
- Fixed a hang when there are more than 24 data files.

- Fixed a crash when `MAX(col)` [884] is selected from an empty table, and `col` is not the first column in a multi-column index.
- Fixed a bug in purge which could cause crashes.

C.3.17 Changes in Release 3.23.43 (04 October 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a bug in `INSERT DELAYED` and `FLUSH TABLES` introduced in 3.23.42.
- Fixed unlikely bug, which returned nonmatching rows, in `SELECT` with many tables and multi-column indexes and 'range' type.
- Fixed an unlikely core dump bug when doing `EXPLAIN SELECT` when using many tables and `ORDER BY`.
- Fixed bug in `LOAD DATA FROM MASTER` when using table with `CHECKSUM=1`.
- Added unique error message when a DEADLOCK occurs during a transaction with `BDB` tables.
- Fixed problem with `BDB` tables and `UNIQUE` columns defined as `NULL`.
- Fixed problem with `myisampack` when using pre-space filled `CHAR` columns.
- Applied patch from Yuri Dario for OS/2.
- Fixed bug in `--safe-user-create` [392].

C.3.18 Changes in Release 3.23.42 (08 September 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem when using `LOCK TABLES` and `BDB` tables.
- Fixed problem with `REPAIR TABLE` on `MyISAM` tables with row lengths in the range from 65517 to 65520 bytes.
- Fixed rare hang when doing `mysqladmin shutdown` when there was a lot of activity in other threads.
- Fixed problem with `INSERT DELAYED` where delayed thread could be hanging on `upgrading locks` for no apparent reason.
- Fixed problem with `myisampack` and `BLOB`.
- Fixed problem when one edited `.MRG` tables by hand. (Patch from Benjamin Pflugmann).
- Enforce that all tables in a `MERGE` table come from the same database.
- Fixed bug with `LOAD DATA INFILE` and transactional tables.
- Fix bug when using `INSERT DELAYED` with wrong column definition.

- Fixed core dump during `REPAIR TABLE` of some particularly broken tables.
- Fixed bug in `InnoDB` and `AUTO_INCREMENT` columns.
- Fixed bug in `InnoDB` and `RENAME TABLE` columns.
- Fixed critical bug in `InnoDB` and `BLOB` columns. If you have used `BLOB` columns larger than 8000 bytes in an `InnoDB` table, it is necessary to dump the table with `mysqldump`, drop it and restore it from the dump.
- Applied large patch for OS/2 from Yuri Dario.
- Fixed problem with `InnoDB` when one could get the error `Can't execute the given command...` even when no transaction was active.
- Applied some minor fixes that concern Gemini.
- Use real arithmetic operations even in integer context if not all arguments are integers. (Fixes uncommon bug in some integer contexts).
- Do not force everything to lowercase on Windows. (To fix problem with Windows and `ALTER TABLE`.) Now `--lower_case_table_names` [418] also works on Unix.
- Fixed that automatic rollback is done when thread end doesn't lock other threads.

C.3.19 Changes in Release 3.23.41 (11 August 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `--sql-mode=value[,value[,value]]` [394] option to `mysqld`. See Section 5.1.2, “Server Command Options”.
- Fixed possible problem with `shutdown` on Solaris where the `.pid` file wasn't deleted.
- `InnoDB` now supports < 4GB rows. The former limit was 8000 bytes.
- The `doublewrite` file flush method is used in `InnoDB`. It reduces the need for Unix `fsync()` calls to a fraction and improves performance on most Unix flavors.
- You can now use the `InnoDB` Monitor to print a lot of `InnoDB` state information, including locks, to the standard output. This is useful in performance tuning.
- Several bugs which could cause hangs in `InnoDB` have been fixed.
- Split `record_buffer` to `record_buffer` and `record_rnd_buffer`. To make things compatible to previous MySQL versions, if `record_rnd_buffer` is not set, then it takes the value of `record_buffer`.
- Fixed optimizing bug in `ORDER BY` where some `ORDER BY` parts where wrongly removed.
- Fixed overflow bug with `ALTER TABLE` and `MERGE` tables.
- Added prototypes for `my_thread_init()` and `my_thread_end()` to `mysql_com.h`
- Added `--safe-user-create` [392] option to `mysqld`.

- Fixed bug in `SELECT DISTINCT ... HAVING` that caused error message `Can't find record in #...`

C.3.20 Changes in Release 3.23.40 (18 July 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem with `--low-priority-updates` [389] and `INSERT` statements.
- Fixed bug in slave thread when under some rare circumstances it could get 22 bytes ahead on the offset in the master.
- Added `slave_net_timeout` [1181] for replication.
- Fixed problem with `UPDATE` and `BDB` tables.
- Fixed hard bug in `BDB` tables when using key parts.
- Fixed problem when using `GRANT FILE ON database.* ...`; previously we added the `DROP` [491] privilege for the database.
- Fixed `DELETE FROM tbl_name ... LIMIT 0` and `UPDATE FROM tbl_name ... LIMIT 0`, which acted as though the `LIMIT` clause was not present (they deleted or updated all selected rows).
- `CHECK TABLE` now checks whether an `AUTO_INCREMENT` column contains the value 0.
- Sending a `SIGHUP` to `mysqld` now only flushes the logs, but does not reset the replication.
- Fixed parser to allow floats of type `1.0e1` (no sign after `e`).
- Option `--force` [316] to `myisamchk` now also updates states.
- Added option `--warnings` [389] to `mysqld`. Now `mysqld` prints the error `Aborted connection` only if this option is used.
- Fixed problem with `SHOW CREATE TABLE` when you didn't have a `PRIMARY KEY`.
- Properly fixed the rename of `innodb_unix_file_flush_method` variable to `innodb_flush_method` [1069].
- Fixed bug when converting `BIGINT UNSIGNED` to `DOUBLE`. This caused a problem when doing comparisons with `BIGINT` values outside of the signed range.
- Fixed bug in `BDB` tables when querying empty tables.
- Fixed a bug when using `COUNT(DISTINCT)` [882] with `LEFT JOIN` and there weren't any matching rows.
- Removed all documentation referring to the `GEMINI` table type. `GEMINI` is not released under an Open Source license.

C.3.21 Changes in Release 3.23.39 (12 June 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please

consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- The `AUTO_INCREMENT` sequence wasn't reset when dropping and adding an `AUTO_INCREMENT` column.
- `CREATE ... SELECT` now creates nonunique indexes delayed.
- Fixed problem where `LOCK TABLES tbl_name READ` followed by `FLUSH TABLES` put an exclusive lock on the table.
- `REAL @variable` values were represented with only 2 digits when converted to strings.
- Fixed problem that client “hung” when `LOAD TABLE FROM MASTER` failed.
- `myisamchk --fast --force` no longer repairs tables that only had the open count wrong.
- Added functions to handle symbolic links to make life easier in 4.0.
- We are now using the `-lcma` thread library on HP-UX 10.20 so that MySQL is more stable on HP-UX.
- Fixed problem with `IF()` [790] and number of decimals in the result.
- Fixed date-part extraction functions to work with dates where day or month is 0.
- Extended argument length in option files from 256 to 512 chars.
- Fixed problem with shutdown when `INSERT DELAYED` was waiting for a `LOCK TABLE`.
- Fixed core dump bug in `InnoDB` when tablespace was full.
- Fixed problem with `MERGE` tables and big tables (larger than 4GB) when using `ORDER BY`.

C.3.22 Changes in Release 3.23.38 (09 May 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a bug when `SELECT` from `MERGE` table sometimes results in incorrectly ordered rows.
- Fixed a bug in `REPLACE()` [800] when using the `ujis` character set.
- Applied Sleepycat `BDB` patches 3.2.9.1 and 3.2.9.2.
- Added `--skip-stack-trace` [394] option to `mysqld`.
- `CREATE TEMPORARY` now works with `InnoDB` tables.
- `InnoDB` now promotes sub keys to whole keys.
- Added option `CONCURRENT` to `LOAD DATA`.
- Better error message when slave `max_allowed_packet` [418] is too low to read a very long log event from the master.
- Fixed bug when too many rows were removed when using `SELECT DISTINCT ... HAVING`.

- `SHOW CREATE TABLE` now returns `TEMPORARY` for temporary tables.
- Added `Rows_examined` to slow query log.
- Fixed problems with function returning empty string when used together with a group function and a `WHERE` that didn't match any rows.
- New program `mysqlcheck`.
- Added database name to output for administrative commands like `CHECK TABLE`, `REPAIR TABLE`, `OPTIMIZE TABLE`.
- Lots of portability fixes for `InnoDB`.
- Changed optimizer so that queries like `SELECT * FROM tbl_name, tbl_name2 ... ORDER BY key_part1 LIMIT row_count` use an index on `key_part1` instead of `filesort`.
- Fixed bug when doing `LOCK TABLE to_table WRITE,...; INSERT INTO to_table... SELECT ...` when `to_table` was empty.
- Fixed bug with `LOCK TABLE` and `BDB` tables.

C.3.23 Changes in Release 3.23.37 (17 April 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a bug when using `MATCH()` [845] in `HAVING` clause.
- Fixed a bug when using `HEAP` tables with `LIKE` [804].
- Added `--mysql-version` option to `safe_mysqld`
- Changed `INNOBASE` to `InnoDB` (because the `INNOBASE` name was in use). All `configure` options and `mysqld` start options now use `innodb` instead of `innobase`. This means that before upgrading to this version, you have to change any configuration files where you have used `innobase` options!
- Fixed bug when using indexes on `CHAR(255) NULL` columns.
- Slave threads now start even if `master-host` is not set, as long as `server-id` is set and valid `master.info` is present.
- Partial updates (terminated with kill) are now logged with a special error code to the binary log. Slave refuses to execute them if the error code indicates the update was terminated abnormally, and has to be recovered with `SET SQL_SLAVE_SKIP_COUNTER=1; SLAVE START` after a manual sanity check/correction of data integrity.
- Fixed bug that erroneously logged a drop of internal temporary table on thread termination to the binary log --- this bug affected replication.
- Fixed a bug in `REGEXP` on 64-bit machines.
- `UPDATE` and `DELETE` with `WHERE unique_key_part IS NULL` didn't update/delete all rows.
- Disabled `INSERT DELAYED` for tables that support transactions.

- Fixed bug when using date functions on `TEXT`/`BLOB` column with wrong date format.
- UDFs now also work on Windows. (Patch by Ralph Mason.)
- Fixed bug in `ALTER TABLE` and `LOAD DATA INFILE` that disabled key-sorting. These commands should now be faster in most cases.
- Fixed performance bug where reopened tables (tables that had been waiting for `FLUSH` or `REPAIR TABLE`) would not use indexes for the next query.
- Fixed problem with `ALTER TABLE` to `InnoDB` tables on FreeBSD.
- Added `mysqld` variables `myisam_max_sort_file_size` [421] and `myisam_max_extra_sort_file_size` [421].
- Initialize signals early to avoid problem with signals in `InnoDB`.
- Applied patch for the `tis620` character set to make comparisons case-independent and to fix a bug in `LIKE` [804] for this character set.

Note

All tables that use the `tis620` character set must be fixed with `myisamchk -r` or `REPAIR TABLE`!

- Added `--skip-safemalloc` [394] option to `mysqld`.

C.3.24 Changes in Release 3.23.36 (27 March 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a bug that permitted use of database names containing a "`.`" character. This fixes a serious security issue when `mysqld` is run as root. (CVE-2001-0407)
- Fixed bug when thread creation failed (could happen when doing a **lot** of connections in a short time).
- Fixed some problems with `FLUSH TABLES` and `TEMPORARY` tables. (Problem with freeing the key cache and error `Can't reopen table...`.)
- Fixed a problem in `InnoDB` with other character sets than `latin1` and another problem when using many columns.
- Fixed bug that caused a core dump when using a very complex query involving `DISTINCT` and summary functions.
- Added the `SET TRANSACTION ISOLATION LEVEL` statement.
- Added `FOR UPDATE` for `SELECT` statements.
- Fixed a bug where the number of affected rows was not returned when MySQL was compiled without transaction support.
- Fixed a bug in `UPDATE` where keys were not always used to find the rows to be updated.
- Fixed a bug in `CONCAT_WS()` [795] where it returned incorrect results.

- Changed `CREATE ... SELECT` and `INSERT ... SELECT` to not allow concurrent inserts as this could make the binary log hard to repeat. (Concurrent inserts are enabled if you are not using the binary or update log.)
- Changed some macros to be able to use fast mutex with `glibc` 2.2.

C.3.25 Changes in Release 3.23.35 (15 March 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed newly introduced bug in `ORDER BY`.
- Fixed wrong define `CLIENT_TRANSACTIONS`.
- Fixed bug in `SHOW VARIABLES` when using `INNOBASE` tables.
- Setting and using user variables in `SELECT DISTINCT` didn't work.
- Tuned `SHOW ANALYZE` for small tables.
- Fixed handling of arguments in the benchmark script `run-all-tests`.

C.3.26 Changes in Release 3.23.34a (11 March 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added extra files to the distribution to allow `INNOBASE` support to be compiled.

C.3.27 Changes in Release 3.23.34 (10 March 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added the `INNOBASE` storage engine and the `BDB` storage engine to the MySQL source distribution.
- Updated the documentation about `GEMINI` tables.
- Fixed a bug in `INSERT DELAYED` that caused threads to hang when inserting `NULL` into an `AUTO_INCREMENT` column.
- Fixed a bug in `CHECK TABLE` / `REPAIR TABLE` that could cause a thread to hang.
- Fixed problem that `REPLACE` would not replace a row that conflicts with an `AUTO_INCREMENT` generated key.
- `mysqld` now only sets `CLIENT_TRANSACTIONS` in `mysql->server_capabilities` if the server supports a transaction-safe storage engine.
- Fixed `LOAD DATA INFILE` to allow numeric values to be read into `ENUM` and `SET` columns.

- Improved error diagnostic for slave thread exit.
- Fixed bug in `ALTER TABLE ... ORDER BY`.
- Added `max_user_connections` [420] variable to `mysqld`.
- Limit query length for replication by `max_allowed_packet` [418], not the arbitrary limit of 4MB.
- Permit space around `=` in argument to `--set-variable`.
- Fixed problem in automatic repair that could leave some threads in state `Waiting for table`.
- `SHOW CREATE TABLE` now displays the `UNION=()` for `MERGE` tables.
- `ALTER TABLE` now remembers the old `UNION=()` definition.
- Fixed bug when replicating timestamps.
- Fixed bug in bidirectional replication.
- Fixed bug in the `BDB` storage engine that occurred when using an index on multiple-part key where a key part may be `NULL`.
- Fixed `MAX()` [884] optimization on sub-key for `BDB` tables.
- Fixed problem where garbage results were returned when using `BDB` tables and `BLOB` or `TEXT` fields when joining many tables.
- Fixed a problem with `BDB` tables and `TEXT` columns.
- Fixed bug when using a `BLOB` key where a const row wasn't found.
- Fixed that `mysqlbinlog` writes the timestamp value for each query. This ensures that one gets same values for date functions like `NOW()` [837] when using `mysqlbinlog` to pipe the queries to another server.
- Permit `--skip-gemini`, `--skip-bdb` [392], and `--skip-innodb` [1066] options to be specified when invoking `mysqld`, even if these storage engines are not compiled in to `mysqld`.
- You can now use `ASC` and `DESC` with `GROUP BY` columns to specify a sort order.
- Fixed a deadlock in the `SET` code, when one ran `SET @foo=bar`, where `bar` is a column reference, an error was not properly generated.

C.3.28 Changes in Release 3.23.33 (09 February 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed DNS lookups not to use the same mutex as the host name cache. This enables known hosts to be quickly resolved even if a DNS lookup takes a long time.
- Added `--character-sets-dir` [329] option to `myisampack`.
- Removed warnings when running `REPAIR TABLE ... EXTENDED`.
- Fixed a bug that caused a core dump when using `GROUP BY` on an alias, where the alias was the same as an existing column name.

- Added `SEQUENCE()` as an example user-defined function.
- Changed `mysql_install_db` to use `BINARY` for `CHAR` columns in the privilege tables.
- Changed `TRUNCATE tbl_name` to `TRUNCATE TABLE tbl_name` to use the same syntax as Oracle. Until 4.0 we also allow `TRUNCATE tbl_name` to not crash old code.
- Fixed “no found rows” bug in `MyISAM` tables when a `BLOB` was first part of a multiple-part key.
- Fixed bug where `CASE` [790] didn't work with `GROUP BY`.
- Added `--sort-recover` [318] option to `myisamchk`.
- `myisamchk -S` and `OPTIMIZE TABLE` now work on Windows.
- Fixed bug when using `DISTINCT` on results from functions that referred to a group function, like:

```
SELECT a, DISTINCT SEC_TO_TIME(SUM(a))
FROM tbl_name GROUP BY a, b;
```

- Fixed buffer overrun in `libmysqlclient` library. Fixed bug in handling `STOP` event after `ROTATE` event in replication.
- Fixed another buffer overrun in `DROP DATABASE`.
- Added `Table_locks_immediate` [457] and `Table_locks_waited` [457] status variables.
- Fixed bug in replication that broke slave server start with existing `master.info`. This fixes a bug introduced in 3.23.32.
- Added `SET SQL_SLAVE_SKIP_COUNTER=n` command to recover from replication glitches without a full database copy.
- Added `max_binlog_size` [1184] variable; the binary log is rotated automatically when the size crosses the limit.
- Added `Last_Error`, `Last_Errno`, and `Slave_skip_counter` variables to `SHOW SLAVE STATUS`.
- Fixed bug in `MASTER_POS_WAIT()` [879] function.
- Execute core dump handler on `SIGILL`, and `SIGBUS` in addition to `SIGSEGV`.
- On x86 Linux, print the current query and thread (connection) id, if available, in the core dump handler.
- Fixed several timing bugs in the test suite.
- Extended `mysqltest` to take care of the timing issues in the test suite.
- `ALTER TABLE` can now be used to change the definition for a `MERGE` table.
- Fixed creation of `MERGE` tables on Windows.
- Portability fixes for OpenBSD and OS/2.
- Added `--temp-pool` [394] option to `mysqld`. Using this option causes most temporary files created to use a small set of names, rather than a unique name for each new file. This is to work around a problem in the Linux kernel dealing with creating a bunch of new files with different names. With the old behavior, Linux seems to "leak" memory, as it is being allocated to the directory entry cache instead of the disk cache.

C.3.29 Changes in Release 3.23.32 (22 January 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Changed code to get around compiler bug in Compaq C++ on OSF/1, that broke `BACKUP TABLE`, `RESTORE TABLE`, `CHECK TABLE`, `REPAIR TABLE`, and `ANALYZE TABLE`.
- Added option `FULL` to `SHOW COLUMNS`. Now we show the privilege list for the columns only if this option is given.
- Fixed bug in `SHOW LOGS` when there weren't any `BDB` logs.
- Fixed a timing problem in replication that could delay sending an update to the client until a new update was done.
- Do not convert field names when using `mysql_list_fields()`. This is to keep this code compatible with `SHOW FIELDS`.
- `MERGE` tables didn't work on Windows.
- Fixed problem with `SET PASSWORD=...` on Windows.
- Added missing `my_config.h` to RPM distribution.
- `TRIM("foo" from "foo")` [803] didn't return an empty string.
- Added `--with-version-suffix` option to `configure`.
- Fixed core dump when client aborted connection without `mysql_close()`.
- Fixed a bug in `RESTORE TABLE` when trying to restore from a nonexistent directory.
- Fixed a bug which caused a core dump on the slave when replicating `SET PASSWORD`.
- Added `MASTER_POS_WAIT()` [879] function.

C.3.30 Changes in Release 3.23.31 (17 January 2001: Production)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- The test suite now tests all reachable `BDB` interface code. During testing we found and fixed many errors in the interface code.
- Using `HAVING` on an empty table could produce one result row when it shouldn't.
- Fixed the MySQL RPM so it no longer depends on Perl5.
- Fixed some problems with `HEAP` tables on Windows.
- `SHOW TABLE STATUS` didn't show correct average row length for tables larger than 4GB.
- `CHECK TABLE ... EXTENDED` didn't check row links for fixed size tables.

- Added option `MEDIUM` to `CHECK TABLE`.
- Fixed problem when using `DECIMAL()` keys on negative numbers.
- `HOUR()` [836] (and some other `TIME` functions) on a `CHAR` column always returned `NULL`.
- Fixed security bug in `SHOW GRANT` (please upgrade if you are using an earlier MySQL 3.23 version). (CVE-2001-1275)
- Fixed buffer overflow bug when writing a certain error message. (CVE-2001-1274)
- Added usage of `setrlimit()` on Linux to get `-O --open_files_limit=`*`val`* to work on Linux.
- Added `bdb_version` [407] variable to `mysqld`.
- Fixed bug when using expression of type:

```
SELECT ... FROM t1 LEFT JOIN t2 ON (t1.a=t2.a) WHERE t1.a=t2.a
```

  In this case the test in the `WHERE` clause was wrongly optimized away.
- Fixed bug in `MyISAM` when deleting keys with possible `NULL` values, but the first key-column was not a prefix-compressed text column.
- Fixed `mysql.server` to read the `[mysql.server]` option file group rather than the `[mysql_server]` group.
- Fixed `safe_mysqld` and `mysql.server` to also read the `server` option section.
- Added `Threads_created` [457] status variable to `mysqld`.

C.3.31 Changes in Release 3.23.30 (04 January 2001)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `SHOW OPEN TABLES` statement.
- Fixed that `myisamdump` works against old `mysqld` servers.
- Fixed `myisamchk -k`*`N`* so that it works again.
- Fixed a problem with replication when the binary log file went over 2G on 32-bit systems.
- `LOCK TABLES` now automatically starts a new transaction.
- Changed `BDB` tables to not use internal subtransactions and reuse open files to get more speed.
- Added `--mysqld=`*`path`* [244] option to `safe_mysqld`.
- Permit hex constants in the `--fields-*-by` and `--lines-terminated-by` options to `mysqldump` and `mysqlimport`.
- Added `--safe-show-database` [391] option to `mysqld`.
- Added `have_bdb` [412], `have_gemini`, `have_innobase`, `have_raid` [413] and `have_openssl` [413] to `SHOW VARIABLES` to make it easy to test for supported extensions.

- Added `--open-files-limit` [391] option to `mysqld`.
- Changed `--open-files` option to `--open-files-limit` in `safe_mysqld`.
- Fixed a bug where some rows were not found with `HEAP` tables that had many keys.
- Fixed that `--bdb-no-sync` [1138] works.
- Changed `--bdb-recover` to `--bdb-no-recover` [1138] as recover should be on by default.
- Changed the default number of `BDB` locks to 10000.
- Fixed a bug from 3.23.29 when allocating the shared structure needed for `BDB` tables.
- Changed `mysqld_multi.sh` to use configure variables. Patch by Christopher McCrory.
- Added fixing of include files for Solaris 2.8.
- Fixed bug with `--skip-networking` [393] on Debian Linux.
- Fixed problem that some temporary files where reported as having the name `UNOPENED` in error messages.
- Fixed bug when running two simultaneous `SHOW LOGS` queries.

C.3.32 Changes in Release 3.23.29 (16 December 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Configure updates for Tru64, large file support, and better TCP wrapper support. By Albert Chin-A-Young.
- Fixed bug in `<=>` operator.
- Fixed bug in `REPLACE` with `BDB` tables.
- `LPAD()` [798] and `RPAD()` [800] shortens the result string if it is longer than the length argument.
- Added `SHOW LOGS` statement.
- Remove unused `BDB` logs on shutdown.
- When creating a table, put `PRIMARY` keys first, followed by `UNIQUE` keys.
- Fixed a bug in `UPDATE` involving multiple-part keys where you specified all key parts both in the update and the `WHERE` part. In this case MySQL could try to update a record that didn't match the whole `WHERE` part.
- Changed drop table to first drop the tables and then the `.frm` file.
- Fixed a bug in the host name cache which caused `mysqld` to report the host name as `''` in some error messages.
- Fixed a bug with `HEAP` type tables; the variable `max_heap_table_size` [419] wasn't used. Now either `MAX_ROWS` or `max_heap_table_size` [419] can be used to limit the size of a `HEAP` type table.

- Changed the default `server-id` value to 1 for masters and 2 for slaves to make it easier to use the binary log.
- Renamed `bdb_lock_max` [407] variable to `bdb_max_lock` [407].
- Added support for `AUTO_INCREMENT` on sub-fields for `BDB` tables.
- Added `ANALYZE TABLE` of `BDB` tables.
- In `BDB` tables, we now store the number of rows; this helps to optimize queries when we need an approximation of the number of rows.
- If we get an error in a multiple-row statement, we now only roll back the last statement, not the entire transaction.
- If you do a `ROLLBACK` when you have updated a nontransactional table you get an error as a warning.
- Added `--bdb-shared-data` [1138] option to `mysqld`.
- Added `Slave_open_temp_tables` [454] status variable to `mysqld`
- Added `binlog_cache_size` [407] and `max_binlog_cache_size` [1183] variables to `mysqld`.
- `DROP TABLE`, `RENAME TABLE`, `CREATE INDEX` and `DROP INDEX` are now transaction endpoints.
- If you do a `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted.
- Fixed `DROP DATABASE` to work on OS/2.
- Fixed bug when doing a `SELECT DISTINCT ... table1 LEFT JOIN table2 ...` when `table2` was empty.
- Added `--abort-slave-event-count` [1173] and `--disconnect-slave-event-count` [1173] options to `mysqld` for debugging and testing of replication.
- Fixed replication of temporary tables. Handles everything except slave server restart.
- `SHOW KEYS` now shows whether key is `FULLTEXT`.
- New script `mysqld_multi`. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.
- Added new script, `mysql-multi.server.sh`. Thanks to Tim Bunce `<Tim.Bunce@ig.co.uk>` for modifying `mysql.server` to easily handle hosts running many `mysqld` processes.
- `safe_mysqld`, `mysql.server`, and `mysql_install_db` have been modified to use `my_print_defaults` instead of various hacks to read the `my.cnf` files. In addition, the handling of various paths has been made more consistent with how `mysqld` handles them by default.
- Automatically remove Berkeley DB transaction logs that no longer are in use.
- Fixed bug with several `FULLTEXT` indexes in one table.
- Added a warning if number of rows changes on `REPAIR TABLE`/`OPTIMIZE TABLE`.
- Applied patches for OS/2 by `Yuri Dario`.
- `FLUSH TABLES` *`tbl_name`* didn't always flush the index tree to disk properly.

- `--bootstrap` [384] is now run in a separate thread. This fixes a problem that caused `mysql_install_db` to core dump on some Linux machines.
- Changed `mi_create()` to use less stack space.
- Fixed bug with optimizer trying to over-optimize `MATCH()` [845] when used with `UNIQUE` key.
- Changed `crash-me` and the MySQL benchmarks to also work with FrontBase.
- Permit `RESTRICT` and `CASCADE` after `DROP TABLE` to make porting easier.
- Reset status variable which could cause problem if one used `--log-slow-queries` [388].
- Added `connect_timeout` [409] variable to `mysql` and `mysqladmin`.
- Added `connect-timeout` as an alias for `timeout` for option files read by `mysql_options()`.

C.3.33 Changes in Release 3.23.28 (22 November 2000: Gamma)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added new options `--pager[=...]` [262], `--no-pager` [262], `--tee=...` [264] and `--no-tee` [264] to the `mysql` client. The new corresponding interactive commands are `pager`, `nopager`, `tee` and `notee`. See Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”, `mysql --help` and the interactive help for more information.
- Fixed crash when automatic repair of `MyISAM` table failed.
- Fixed a major performance bug in the table locking code when a lot of `SELECT`, `UPDATE` and `INSERT` statements constantly were running. The symptom was that the `UPDATE` and `INSERT` queries were locked for a long time while new `SELECT` statements were executed before the updates.
- When reading `options_files` with `mysql_options()` the `return-found-rows` option was ignored.
- You can now specify `interactive-timeout` in the option file that is read by `mysql_options()`. This makes it possible to force programs that run for a long time (like `mysqlhotcopy`) to use the `interactive_timeout` [414] time instead of the `wait_timeout` [434] time.
- Added to the slow query log the time and the user name for each logged query. If you are using `--log-long-format` [388] then also queries that do not use an index are logged, even if the query takes less than `long_query_time` [417] seconds.
- Fixed a problem in `LEFT JOIN` which caused all columns in a reference table to be `NULL`.
- Fixed a problem when using `NATURAL JOIN` without keys.
- Fixed a bug when using a multiple-part keys where the first part was of type `TEXT` or `BLOB`.
- `DROP` of temporary tables wasn't stored in the update log or binary log.
- Fixed a bug where `SELECT DISTINCT * ... LIMIT row_count` only returned one row.
- Fixed a bug in the assembler code in `strstr()` for SPARC and cleaned up the `global.h` header file to avoid a problem with bad aliasing with the compiler submitted with Red Hat 7.0. (Reported by Trond Eivind Glomsrød)

- The `--skip-networking` [393] option now works properly on NT.
- Fixed a long outstanding bug in the `ISAM` tables when a row with a length of more than 65KB was shortened by a single byte.
- Fixed a bug in `MyISAM` when running multiple updating processes on the same table.
- Permit `FLUSH TABLES` with a *`tbl_name`* option.
- Added `--replicate-ignore-table` [1177], `--replicate-do-table` [1177], `--replicate-wild-ignore-table` [1178], and `--replicate-wild-do-table` [1178] options to `mysqld`.
- Changed all log files to use our own `IO_CACHE` mechanism instead of `FILE` to avoid OS problems when there are many files open.
- Added `--open-files` and `--timezone` options to `safe_mysqld`.
- Fixed a fatal bug in `CREATE TEMPORARY TABLE ... SELECT ...`.
- Fixed a problem with `CREATE TABLE ... SELECT NULL`.
- Added variables `large_file_support`,`net_read_timeout` [422], `net_write_timeout` [422] and `query_buffer_size` to `SHOW VARIABLES`.
- Added status variables `Created_tmp_files` [451] and `Sort_merge_passes` [455] to `SHOW STATUS`.
- Fixed a bug where we didn't allow an index name after the `FOREIGN KEY` definition.
- Added `TRUNCATE` *`tbl_name`* as a synonym for `DELETE FROM` *`tbl_name`*.
- Fixed a bug in a `BDB` key compare function when comparing part keys.
- Added `bdb_lock_max` [407] variable to `mysqld`.
- Added more tests to the benchmark suite.
- Fixed an overflow bug in the client code when using overly long database names.
- `mysql_connect()` now aborts on Linux if the server doesn't answer in `timeout` seconds.
- `SLAVE START` did not work if you started with `--skip-slave-start` [1179] and had not explicitly run `CHANGE MASTER TO`.
- Fixed the output of `SHOW MASTER STATUS` to be consistent with `SHOW SLAVE STATUS`. (It now has no directory in the log name.)
- Added `PURGE BINARY LOGS TO`.
- Added `SHOW MASTER LOGS` statement to display a list of binary log files.
- Added `--safemalloc-mem-limit` option to `mysqld` to simulate memory shortage when compiled with the `--with-debug=full` [98] option.
- Fixed several core dumps in out-of-memory conditions.
- `SHOW SLAVE STATUS` was using an uninitialized mutex if the slave had not been started yet.
- Fixed bug in `ELT()` [795] and `MAKE_SET()` [799] when the query used a temporary table.
- `CHANGE MASTER TO` without specifying `MASTER_LOG_POS` would set it to 0 instead of 4 and hit the magic number in the master binary log.

- `ALTER TABLE ... ORDER BY ...` syntax added. This creates the new table with the rows in a specific order.

C.3.34 Changes in Release 3.23.27 (24 October 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a bug where the automatic repair of `MyISAM` tables sometimes failed when the data file was corrupt.
- Fixed a bug in `SHOW CREATE` when using `AUTO_INCREMENT` columns.
- Changed `BDB` tables to use new compare function in Berkeley DB 3.2.3.
- You can now use Unix socket files with MIT-pthreads.
- Added the `latin5` (turkish) character set.
- Small portability fixes.

C.3.35 Changes in Release 3.23.26 (18 October 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Renamed `FLUSH MASTER` and `FLUSH SLAVE` to `RESET MASTER` and `RESET SLAVE`.
- Fixed `<>` to work properly with `NULL`.
- Fixed a problem with `SUBSTRING_INDEX()` [802] and `REPLACE()` [800]. (Patch by Alexander Igonitchev)
- Fix `CREATE TEMPORARY TABLE IF NOT EXISTS` not to produce an error if the table exists.
- If you don't create a `PRIMARY KEY` in a `BDB` table, a hidden `PRIMARY KEY` is created.
- Added read-only-key optimization to `BDB` tables.
- `LEFT JOIN` in some cases preferred a full table scan when there was no `WHERE` clause.
- When using `--log-slow-queries` [388], don't count the time waiting for a lock.
- Fixed bug in lock code on Windows which could cause the key cache to report that the key file was crashed even if it was okay.
- Automatic repair of `MyISAM` tables if you start `mysqld` with `--myisam-recover` [390].
- Removed the `TYPE=` keyword from `CHECK TABLE` and `REPAIR TABLE`. Permit `CHECK TABLE` options to be combined. (You can still use `TYPE=`, but this usage is deprecated.)
- Fixed mutex bug in the binary replication log --- long update queries could be read only in part by the slave if it did it at the wrong time, which was not fatal, but resulted in a performance-degrading reconnect and a scary message in the error log.

- Changed the format of the binary log --- added magic number, server version, binary log version. Added the server ID and query error code for each query event.
- Replication thread from the slave now kills all the stale threads from the same server.
- Long replication user names were not being handled properly.
- Added `--replicate-rewrite-db` [1177] option to `mysqld`.
- Added `--skip-slave-start` [1179] option to `mysqld`.
- Updates that generated an error code (such as `INSERT INTO foo(some_key) values (1),(1)`) erroneously terminated the slave thread.
- Added optimization of queries where `DISTINCT` is used only on columns from some of the tables.
- Permit floating-point numbers where there is no sign after the exponent (like `1e1`).
- `SHOW GRANTS` didn't always show all column grants.
- Added `--defaults-extra-file=`*`file_name`* [235] option to all MySQL clients.
- Columns referenced in `INSERT` statements now are initialized properly.
- `UPDATE` didn't always work when used with a range on a timestamp that was part of the key that was used to find rows.
- Fixed a bug in `FULLTEXT` index when inserting a `NULL` column.
- Changed to use `mkstemp()` instead of `tempnam()`. Based on a patch from John Jones.

C.3.36 Changes in Release 3.23.25 (29 September 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed that `databasename` works as second argument to `mysqlhotcopy`.
- The values for the `UMASK` and `UMASK_DIR` environment variables now can be specified in octal by beginning the value with a zero.
- Added `RIGHT JOIN`. This makes `RIGHT` a reserved word.
- Added `@@identity` as a synonym for `LAST_INSERT_ID()` [874]. (This is for MSSQL compatibility.)
- Fixed a bug in `myisamchk` and `REPAIR TABLE` when using `FULLTEXT` index.
- `LOAD DATA INFILE` now works with FIFOs. (Patch by Toni L. Harbaugh-Blackford.)
- `FLUSH LOGS` broke replication if you specified a log name with an explicit extension as the value of the `log-bin` option.
- Fixed a bug in `MyISAM` with packed multiple-part keys.
- Fixed crash when using `CHECK TABLE` on Windows.
- Fixed a bug where `FULLTEXT` index always used the `koi8_ukr` character set.

- Fixed privilege checking for `CHECK TABLE`.
- The `MyISAM` repair/reindex code didn't use the `--tmpdir` [394] option for its temporary files.
- Added `BACKUP TABLE` and `RESTORE TABLE`.
- Fixed core dump on `CHANGE MASTER TO` when the slave did not have the master to start with.
- Fixed incorrect `Time` in the processlist for `Connect` of the slave thread.
- The slave now logs when it connects to the master.
- Fixed a core dump bug when doing `FLUSH MASTER` if you didn't specify a file name argument to `--log-bin` [1181].
- Added missing `ha_berkeley.x` files to the MySQL Windows distribution.
- Fixed some mutex bugs in the log code that could cause thread blocks if new log files couldn't be created.
- Added lock time and number of selected processed rows to slow query log.
- Added `--memlock` [389] option to `mysqld` to lock `mysqld` in memory on systems with the `mlockall()` call (as in Solaris).
- `HEAP` tables didn't use keys properly. (Bug from 3.23.23.)
- Added better support for `MERGE` tables (keys, mapping, creation, documentation...). See Section 13.3, “The `MERGE` Storage Engine”.
- Fixed bug in `mysqldump` from 3.23 which caused some `CHAR` columns not to be quoted.
- Merged `analyze`, `check`, `optimize` and repair code.
- `OPTIMIZE TABLE` is now mapped to `REPAIR TABLE` with statistics and sorting of the index tree. This means that for the moment it only works on `MyISAM` tables.
- Added a pre-alloced block to root_malloc to get fewer mallocs.
- Added a lot of new statistics variables.
- Fixed `ORDER BY` bug with `BDB` tables.
- Removed warning that `mysqld` couldn't remove the `.pid` file under Windows.
- Changed `--log-isam` [388] to log `MyISAM` tables instead of isam tables.
- Fixed `CHECK TABLE` to work on Windows.
- Added file mutexes to make `pwrite()` safe on Windows.

C.3.37 Changes in Release 3.23.24 (08 September 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `Created_tmp_disk_tables` [450] variable to `mysqld`.

- To make it possible to reliably dump and restore tables with `TIMESTAMP(X)` columns, MySQL now reports columns with *`X`* other than 14 or 8 to be strings.
- Changed sort order for `latin1` as it was before MySQL 3.23.23. Any table that was created or modified with 3.23.22 must be repaired if it has `CHAR` columns that may contain characters with ASCII values greater than 128!
- Fixed small memory leak introduced from 3.23.22 when creating a temporary table.
- Fixed problem with `BDB` tables and reading on a unique (not primary) key.
- Restored the `win1251` character set (it is now only marked deprecated).

C.3.38 Changes in Release 3.23.23 (01 September 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Changed sort order for 'German'; all tables created with 'German' sortorder must be repaired with `REPAIR TABLE` or `myisamchk` before use!
- Added `--core-file` [385] option to `mysqld` to get a core file on Linux if `mysqld` dies on the `SIGSEGV` signal.
- MySQL client `mysql` now starts with option `--no-named-commands` [261] (`-g`) by default. This option can be disabled with `--enable-named-commands` [261] (`-G`). This may cause incompatibility problems in some cases, for example, in SQL scripts that use named commands without a semicolon! Long format commands still work from the first line.
- Fixed a problem when using many pending `DROP TABLE` statements at the same time.
- Optimizer didn't use keys properly when using `LEFT JOIN` on an empty table.
- Added shorter help text when invoking `mysqld` with incorrect options.
- Fixed nonfatal `free()` bug in `mysqlimport`.
- Fixed bug in `MyISAM` index handling of `DECIMAL`/`NUMERIC` keys.
- Fixed a bug in concurrent insert in `MyISAM` tables. In some contexts, usage of `MIN(key_part)` [884] or `MAX(key_part)` [884] returned an empty set.
- Updated `mysqlhotcopy` to use the new `FLUSH TABLES table_list` syntax. Only tables which are being backed up are flushed now.
- Changed behavior of `--enable-thread-safe-client` [98] so that both nonthreaded (`-lmysqlclient`) and threaded (`-lmysqlclient_r`) libraries are built. Users who linked against a threaded `-lmysqlclient` need to link against `-lmysqlclient_r` now.
- Added atomic `RENAME TABLE` command.
- Do not count `NULL` values in `COUNT(DISTINCT ...)` [882].
- Changed `ALTER TABLE`, `LOAD DATA INFILE` on empty tables and `INSERT ... SELECT ...` on empty tables to create nonunique indexes in a separate batch with sorting. This makes these statements much faster when you have many indexes.

- `ALTER TABLE` now logs the first used insert_id correctly.
- Fixed crash when adding a default value to a `BLOB` column.
- Fixed a bug with `DATE_ADD/DATE_SUB` where it returned a datetime instead of a date.
- Fixed a problem with the thread cache which made some threads show up as `***DEAD***` in `SHOW PROCESSLIST`.
- Fixed a lock in our thr_rwlock code, which could make selects that run at the same time as concurrent inserts crash. This affects only systems that don't have the `pthread_rwlock_rdlock` code.
- When deleting rows with a nonunique key in a `HEAP` table, all rows weren't always deleted.
- Fixed bug in range optimizer for `HEAP` tables for searches on a part index.
- Fixed `SELECT` on part keys to work with `BDB` tables.
- Fixed `INSERT INTO bdb_table ... SELECT` to work with `BDB` tables.
- `CHECK TABLE` now updates key statistics for the table.
- `ANALYZE TABLE` now only updates tables that have been changed since the last `ANALYZE TABLE`. Note that this is a new feature and tables are not marked to be analyzed until they are updated in any way with 3.23.23 or newer. For older tables, you have to do `CHECK TABLE` to update the key distribution.
- Fixed some minor privilege problems with `CHECK TABLE`, `ANALYZE TABLE`, `REPAIR TABLE` and `SHOW CREATE` statements.
- Added `CHANGE MASTER TO` statement.
- Added `FAST`, `QUICK EXTENDED` check types to `CHECK TABLES`.
- Changed `myisamchk` so that `--fast` [316] and `--check-only-changed` [316] are also honored with `--sort-index` [319] and `--analyze` [318].
- Fixed fatal bug in `LOAD TABLE FROM MASTER` that did not lock the table during index re-build.
- `LOAD DATA INFILE` broke replication if the database was excluded from replication.
- More variables in `SHOW SLAVE STATUS` and `SHOW MASTER STATUS`.
- `SLAVE STOP` now does not return until the slave thread actually exits.
- Full-text search using the `MATCH()` [845] function and `FULLTEXT` index type (for `MyISAM` files). This makes `FULLTEXT` a reserved word.

C.3.39 Changes in Release 3.23.22 (31 July 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed that `lex_hash.h` is created properly for each MySQL distribution.
- Fixed that `MASTER` and `COLLECTION` are not reserved words.

- The log generated by `--log-slow-queries` [388] didn't contain the whole queries.
- Fixed that open transactions in `BDB` tables are rolled back if the connection is closed unexpectedly.
- Added workaround for a bug in `gcc` 2.96 (intel) and `gcc` 2.9 (IA-64) in `gen_lex_hash.c`.
- Fixed memory leak in the client library when using `host=` in the `my.cnf` file.
- Optimized functions that manipulate the hours/minutes/seconds.
- Fixed bug when comparing the result of `DATE_ADD()` [829]/`DATE_SUB()` [833] against a number.
- Changed the meaning of `-F`, `--fast` [316] for `myisamchk`. Added `-C`, `--check-only-changed` [316] option to `myisamchk`.
- Added `ANALYZE tbl_name` to update key statistics for tables.
- Changed binary items `0x...` to be regarded as integers by default.
- Fix for SCO and `SHOW PROCESSLIST`.
- Added `auto-rehash` on reconnect for the `mysql` client.
- Fixed a newly introduced bug in `MyISAM`, where the index file couldn't get bigger than 64MB.
- Added `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`.

C.3.40 Changes in Release 3.23.21 (04 July 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `mysql_character_set_name()` function to the MySQL C API.
- Made the update log ASCII 0 safe.
- Added the `mysql_config` script.
- Fixed problem when using `<` or `>` with a char column that was only partly indexed.
- One would get a core dump if the log file was not readable by the MySQL user.
- Changed `mysqladmin` to use `CREATE DATABASE` and `DROP DATABASE` statements instead of the old deprecated API calls.
- Fixed `chown` warning in `safe_mysqld`.
- Fixed a bug in `ORDER BY` that was introduced in 3.23.19.
- Only optimize the `DELETE FROM tbl_name` to do a drop+create of the table if we are in `autocommit` [406] mode (needed for `BDB` tables).
- Added extra checks to avoid index corruption when the `ISAM`/`MyISAM` index files get full during an `INSERT`/`UPDATE`.
- `myisamchk` didn't correctly update row checksum when used with `-ro` (this only gave a warning in subsequent runs).

- Fixed bug in `REPAIR TABLE` so that it works with tables without indexes.
- Fixed buffer overrun in `DROP DATABASE`.
- `LOAD TABLE FROM MASTER` is sufficiently bug-free to announce it as a feature.
- `MATCH` and `AGAINST` are now reserved words.

C.3.41 Changes in Release 3.23.20 (28 June 2000: Beta)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed bug in 3.23.19; `DELETE FROM tbl_name` removed the `.frm` file.
- Added `SHOW CREATE TABLE`.

C.3.42 Changes in Release 3.23.19

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Changed copyright for all files to GPL for the server code and utilities and to LGPL for the client libraries. See http://www.fsf.org/licenses/.
- Fixed bug where all rows matching weren't updated on a `MyISAM` table when doing update based on key on a table with many keys and some key changed values.
- The Linux MySQL RPMs and binaries are now statically linked with a linuxthread version that has faster mutex handling when used with MySQL.
- `ORDER BY` can now use `REF` keys to find subsets of the rows that need to be sorted.
- Changed name of `print_defaults` program to `my_print_defaults` to avoid name confusion.
- Fixed `NULLIF()` [791] to work as required by standard SQL.
- Added `net_read_timeout` [422] and `net_write_timeout` [422] as startup parameters to `mysqld`.
- Fixed bug that destroyed index when doing `myisamchk --sort-records` on a table with prefix compressed index.
- Added `pack_isam` and `myisampack` to the standard MySQL distribution.
- Added the syntax `BEGIN WORK` (the same as `BEGIN`).
- Fixed core dump bug when using `ORDER BY` on a `CONV()` [818] expression.
- Added `LOAD TABLE FROM MASTER`.
- Added `FLUSH MASTER` and `FLUSH SLAVE`.
- Fixed big/little endian problem in the replication.

C.3.43 Changes in Release 3.23.18 (11 June 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a problem from 3.23.17 when choosing character set on the client side.
- Added `FLUSH TABLES WITH READ LOCK` to make a global lock suitable for making a copy of MySQL data files.
- `CREATE TABLE ... SELECT ... PROCEDURE` now works.
- Internal temporary tables now use compressed index when using `GROUP BY` on `VARCHAR/CHAR` columns.
- Fixed a problem when locking the same table with both a `READ` and a `WRITE` lock.
- Fixed problem with `myisamchk` and `RAID` tables.

C.3.44 Changes in Release 3.23.17 (07 June 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed a bug in `FIND_IN_SET()` [796] when the first argument was `NULL`.
- Added table locks to Berkeley DB.
- Fixed a bug with `LEFT JOIN` and `ORDER BY` where the first table had only one matching row.
- Added 4 sample `my.cnf` example files in the `support-files` directory.
- Fixed `duplicated key` problem when doing big `GROUP BY` operations. (This bug was probably introduced in 3.23.15.)
- Changed syntax for `INNER JOIN` to match standard SQL.
- Added `NATURAL JOIN` syntax.
- A lot of fixes in the `BDB` interface.
- Added handling of `--no-defaults` [235] and `--defaults-file` [235] to `safe_mysqld.sh` and `mysql_install_db.sh`.
- Fixed bug in reading compressed tables with many threads.
- Fixed that `USE INDEX` works with `PRIMARY` keys.
- Added `BEGIN` statement to start a transaction in `autocommit` [406] mode.
- Added support for symbolic links for Windows.
- Changed protocol to let client know if the server is in `autocommit` [406] mode and if there is a pending transaction. If there is a pending transaction, the client library gives an error before reconnecting to the

server to let the client know that the server did a rollback. The protocol is still backward-compatible with old clients.

- `KILL` now works on a thread that is locked on a 'write' to a dead client.
- Fixed memory leak in the replication slave thread.
- Added new `log-slave-updates` option to `mysqld`, to allow daisy-chaining the slaves.
- Fixed compile error on FreeBSD and other systems where `pthread_t` is not the same as `int`.
- Fixed master shutdown aborting the slave thread.
- Fixed a race condition in `INSERT DELAYED` code when doing `ALTER TABLE`.
- Added deadlock detection sanity checks to `INSERT DELAYED`.

C.3.45 Changes in Release 3.23.16 (16 May 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `SLAVE START` and `SLAVE STOP` statements.
- Added `TYPE=QUICK` option to `CHECK TABLE` and to `REPAIR TABLE`.
- Fixed bug in `REPAIR TABLE` when the table was in use by other threads.
- Added a thread cache to make it possible to debug MySQL with `gdb` when one does a lot of reconnects. This also improves systems where you can't use persistent connections.
- Lots of fixes in the Berkeley DB interface.
- `UPDATE IGNORE` does not abort if an update results in a `DUPLICATE_KEY` error.
- Put `CREATE TEMPORARY TABLE` statements in the update log.
- Fixed bug in handling of masked IP addresses in the privilege tables.
- Fixed bug with `delay_key_write` [410] tables and `CHECK TABLE`.
- Added `--replicate-do-db` [1176] and `--replicate-ignore-db` [1176] options to `mysqld`, to restrict which databases get replicated.
- Added `sql_log_bin` [429] option.

C.3.46 Changes in Release 3.23.15 (08 May 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- To start `mysqld` as `root`, you must now use the `--user=root` [395] option.
- Added interface to Berkeley DB. (This is not yet functional; play with it at your own risk!)

- Replication between master and slaves.
- Fixed bug that other threads could steal a lock when a thread had a lock on a table and did a `FLUSH TABLES` command.
- Added the `slow_launch_time` [427] variable and the `Slow_launch_threads` [455] status variable to `mysqld`. These can be examined with `mysqladmin variables` and `mysqladmin extended-status`.
- Added functions `INET_NTOA()` [878] and `INET_ATON()` [878].
- The default type of `IF()` [790] now depends on the second and third arguments and not only on the second argument.
- Fixed case when `myisamchk` could go into a loop when trying to repair a crashed table.
- Do not write `INSERT DELAYED` to update log if `sql_log_update = 0` [429].
- Fixed problem with `REPLACE` on `HEAP` tables.
- Added possible character sets and time zone to `SHOW VARIABLES` output.
- Fixed bug in locking code that could result in locking problems with concurrent inserts under high load.
- Fixed a problem with `DELETE` of many rows on a table with compressed keys where MySQL scanned the index to find the rows.
- Fixed problem with `CHECK TABLE` on table with deleted keyblocks.
- Fixed a bug in reconnect (at the client side) where it didn't free memory properly in some contexts.
- Fixed problems in update log when using `LAST_INSERT_ID()` [874] to update a table with an `AUTO_INCREMENT` key.
- Added `NULLIF()` [791] function.
- Fixed bug when using `LOAD DATA INFILE` on a table with `BLOB/TEXT` columns.
- Optimized `MyISAM` to be faster when inserting keys in sorted order.
- `EXPLAIN SELECT ...` now also prints out whether MySQL needs to create a temporary table or use file sorting when resolving the `SELECT`.
- Added optimization to skip `ORDER BY` parts where the part is a constant expression in the `WHERE` part. Indexes can now be used even if the `ORDER BY` doesn't match the index exactly, as long as all the unused index parts and all the extra `ORDER BY` columns are constants in the `WHERE` clause. See Section 7.4.3, “How MySQL Uses Indexes”.
- `UPDATE` and `DELETE` on a whole unique key in the `WHERE` part are now faster than before.
- Changed `RAID_CHUNKSIZE` to be in 1024-byte increments.
- Fixed core dump in `LOAD_FILE(NULL)` [798].

C.3.47 Changes in Release 3.23.14 (09 April 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please

consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `mysqlbinlog` program for displaying binary log files in text format.
- Added `mysql_real_escape_string()` function to the MySQL C API.
- Fixed a bug in `CONCAT()` [795] where one of the arguments was a function that returned a modified argument.
- Fixed a critical bug in `myisamchk`, where it updated the header in the index file when one only checked the table. This confused the `mysqld` daemon if it updated the same table at the same time. Now the status in the index file is only updated if one uses `--update-state` [316]. With older `myisamchk` versions you should use `--read-only` [316] when only checking tables, if there is the slightest chance that the `mysqld` server is working on the table at the same time!
- Fixed that `DROP TABLE` is logged in the update log.
- Fixed problem when searching on `DECIMAL()` key field where the column data contained leading zeros.
- Fix bug in `myisamchk` when the `AUTO_INCREMENT` column isn't the first key.
- Permit `DATETIME` in ISO8601 format: 2000-03-12T12:00:00
- Dynamic character sets. A `mysqld` binary can now handle many different character sets (you can choose which when starting `mysqld`).
- Added `REPAIR TABLE` statement.
- Added `mysql_thread_safe()` function to the MySQL C API.
- Added the `UMASK_DIR` environment variable.
- Added `CONNECTION_ID()` [872] function to return the client connection thread ID.
- When using `=` on `BLOB` or `VARCHAR BINARY` keys, where only a part of the column was indexed, the whole column of the result row wasn't compared.
- Fix for `sjis` character set and `ORDER BY`.
- When running in ANSI mode, don't allow columns to be used that aren't in the `GROUP BY` part.

C.3.48 Changes in Release 3.23.13 (14 March 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem when doing locks on the same table more than 2 times in the same `LOCK TABLE` statement; this fixed the problem one got when running the test-ATIS test with `--fast` or `--check-only-changed`.
- Added `SQL_BUFFER_RESULT` option to `SELECT`.
- Removed endspace from double/float numbers in results from temporary tables.
- Added `CHECK TABLE` statement.

- Added changes for `MyISAM` in 3.23.12 that didn't get into the source distribution because of CVS problems.
- Fixed bug so that `mysqladmin shutdown` waits for the local server to close down.
- Fixed a possible endless loop when calculating timestamp.
- Added `print_defaults` program to the `.rpm` files. Removed `mysqlbug` from the client `.rpm` file.

C.3.49 Changes in Release 3.23.12 (07 March 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed bug in `MyISAM` involving `REPLACE ... SELECT ...` which could give a corrupted table.
- Fixed bug in `myisamchk` where it incorrectly reset the `AUTO_INCREMENT` value.
- LOTS of patches for Linux Alpha. MySQL now appears to be relatively stable on Alpha.
- Changed `DISTINCT` on `HEAP` temporary tables to use hashed keys to quickly find duplicated rows. This mostly concerns queries of type `SELECT DISTINCT ... GROUP BY ...`. This fixes a problem where not all duplicates were removed in queries of the above type. In addition, the new code is MUCH faster.
- Added patches to make MySQL compile on Mac OS X.
- Added `IF NOT EXISTS` clause to `CREATE DATABASE`.
- Added `--all-databases` [293] and `--databases` [294] options to `mysqldump` to allow dumping of many databases at the same time.
- Fixed bug in compressed `DECIMAL()` index in `MyISAM` tables.
- Fixed bug when storing 0 into a timestamp.
- When doing `mysqladmin shutdown` on a local connection, `mysqladmin` now waits until the PID file is gone before terminating.
- Fixed core dump with some `COUNT(DISTINCT ...)` [882] queries.
- Fixed that `myisamchk` works properly with RAID tables.
- Fixed problem with `LEFT JOIN` and *`key_col`* `IS NULL`.
- Fixed bug in `net_clear()` which could give the error `Aborted connection` in the MySQL clients.
- Added options `USE INDEX (`*`index_list`*`)` and `IGNORE INDEX (`*`index_list`*`)` as parameters in `SELECT`.
- `DELETE` and `RENAME` should now work on `RAID` tables.

C.3.50 Changes in Release 3.23.11 (16 February 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please

consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added `HIGH_PRIORITY` option to `INSERT`. This overrides the effect of the `--low-priority-updates` [389] server option and does not perform concurrent inserts.
- Permit the `ALTER TABLE tbl_name ADD (field_list)` syntax.
- Fixed problem with optimizer that could sometimes use incorrect keys.
- Fixed that `GRANT/REVOKE ALL PRIVILEGES` doesn't affect `GRANT OPTION` [491].
- Removed extra "`)`" from the output of `SHOW GRANTS`.
- Fixed problem when storing numbers in timestamps.
- Fix problem with time zones that have half hour offsets.
- Permit the syntax `UNIQUE INDEX` in `CREATE` statements.
- `mysqlhotcopy` - fast online hot-backup utility for local MySQL databases. By Tim Bunce.
- New more secure `mysqlaccess`. Thanks to Steve Harvey for this.
- Added `--i-am-a-dummy` [264] and `--safe-updates` [264] options to `mysql`.
- Added `select_limit` and `max_join_size` [419] variables to `mysql`.
- Added `sql_max_join_size` and `sql_safe_updates` [430] options.
- Added `READ LOCAL` lock that doesn't lock the table for concurrent inserts. (This is used by `mysqldump`.)
- Changed that `LOCK TABLES ... READ` no longer permits concurrent inserts.
- Added `--skip-delay-key-write` option to `mysqld`.
- Fixed security problem in the protocol regarding password checking.
- `_rowid` can now be used as an alias for an integer type unique indexed column.
- Added back blocking of `SIGPIPE` when compiling with `--thread-safe-clients` to make things safe for old clients.

C.3.51 Changes in Release 3.23.10 (30 January 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed bug in 3.23.9 where memory wasn't properly freed when using `LOCK TABLES`.

C.3.52 Changes in Release 3.23.9 (29 January 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem that affected queries that did arithmetic on group functions.
- Fixed problem with timestamps and `INSERT DELAYED`.
- Fixed that `date_col BETWEEN const_date AND const_date` works.
- Fixed problem when only changing a 0 to `NULL` in a table with `BLOB/TEXT` columns.
- Fixed bug in range optimizer when using many key parts and or on the middle key parts: `WHERE K1=1 and K3=2 and (K2=2 and K4=4 or K2=3 and K4=5)`
- Added `source` command to `mysql` to allow reading of batch files inside the `mysql` client. Original patch by Matthew Vanecek.
- Fixed critical problem with the `WITH GRANT OPTION` option.
- Do not give an unnecessary `GRANT` error when using tables from many databases in the same query.
- Added VIO wrapper (needed for SSL support; by Andrei Errapart and Tõnu Samuel).
- Fixed optimizer problem on `SELECT` when using many overlapping indexes. MySQL should now be able to choose keys even better when there are many keys to choose from.
- Changed optimizer to prefer a range key instead of a ref key when the range key can uses more columns than the ref key (which only can use columns with `=`). For example, the following type of queries should now be faster: `SELECT * from key_part_1=const and key_part_2 > const2`
- Fixed bug that a change of all `VARCHAR` columns to `CHAR` columns didn't change row type from dynamic to fixed.
- Disabled floating-point exceptions for FreeBSD to fix core dump when doing `SELECT FLOOR(POW(2,63))`.
- Renamed `mysqld` startup option from `--delay-key-write` to `--delay-key-write-for-all-tables` [386].
- Added `read-next-on-key` to `HEAP` tables. This should fix all problems with `HEAP` tables when using non-`UNIQUE` keys.
- Added option to print default arguments to all clients.
- Added `--log-slow-queries` [388] option to `mysqld` to log all queries that take a long time to a separate log file with a time indicating how long the query took.
- Fixed core dump when doing `WHERE key_col=RAND(...)`.
- Fixed optimization bug in `SELECT ... LEFT JOIN ... key_col IS NULL`, when `key_col` could contain `NULL` values.
- Fixed problem with 8-bit characters as separators in `LOAD DATA INFILE`.

C.3.53 Changes in Release 3.23.8 (02 January 2000)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed problem when handling indexfiles larger than 8GB.
- Added latest patches to MIT-pthreads for NetBSD.
- Fixed problem with time zones that are < GMT - 11.
- Fixed a bug when deleting packed keys in `NISAM`.
- Fixed problem with `ISAM` when doing some `ORDER BY ... DESC` queries.
- Fixed bug when doing a join on a text key which didn't cover the whole key.
- Option `--delay-key-write` [386] didn't enable delayed key writing.
- Fixed update of `TEXT` column which involved only case changes.
- Fixed that `INSERT DELAYED` doesn't update timestamps that are given.
- Added function `YEARWEEK()` [844] and options `x`, `X`, `v` and `V` to `DATE_FORMAT()` [832].
- Fixed problem with `MAX(indexed_column)` [884] and `HEAP` tables.
- Fixed problem with `BLOB NULL` keys and `LIKE "prefix%"` [804].
- Fixed problem with `MyISAM` and fixed-length rows < 5 bytes.
- Fixed problem that could cause MySQL to touch freed memory when doing very complicated `GROUP BY` queries.
- Fixed core dump if you got a crashed table where an `ENUM` field value was too big.

C.3.54 Changes in Release 3.23.7 (10 December 1999)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed workaround under Linux to avoid problems with `pthread_mutex_timedwait()`, which is used with `INSERT DELAYED`. See Section 2.12.1, “Linux Notes”.
- Fixed that one get a 'disk full' error message if one gets disk full when doing sorting (instead of waiting until we got more disk space).
- Fixed a bug in `MyISAM` with keys > 250 characters.
- In `MyISAM` one can now do an `INSERT` at the same time as other threads are reading from the table.
- Added `max_write_lock_count` [421] variable to `mysqld` to force a `READ` lock after a certain number of `WRITE` locks.
- Inverted flag `delay_key_write` [410] on `show variables`.
- Renamed `concurrency` variable to `thread_concurrency` [431].
- The following functions are now multi-byte-safe: `LOCATE(substr,str)` [798], `POSITION(substr IN str)` [799], `LOCATE(substr,str,pos)` [798], `INSTR(str,substr)` [797], `LEFT(str,len)` [797], `RIGHT(str,len)` [800], `SUBSTRING(str,pos,len)` [802], `SUBSTRING(str FROM pos FOR`

len) [802], MID(*str*,*pos*,*len*) [799], SUBSTRING(*str*,*pos*) [802], SUBSTRING(*str* FROM *pos*) [802], SUBSTRING_INDEX(*str*,*delim*,*count*) [802], RTRIM(*str*) [801], TRIM([[BOTH | TRAILING] [*remstr*] FROM] *str*) [803], REPLACE(*str*,*from_str*,*to_str*) [800], REVERSE(*str*) [800], INSERT(*str*,*pos*,*len*,*newstr*) [797], LCASE(*str*) [797], LOWER(*str*) [798], UCASE(*str*) [803] and UPPER(*str*) [804]; patch by Wei He.

- Fix core dump when releasing a lock from a nonexistent table.
- Remove locks on tables before starting to remove duplicates.
- Added option FULL to SHOW PROCESSLIST.
- Added option --verbose [281] to mysqladmin.
- Fixed problem when automatically converting HEAP to MyISAM.
- Fixed bug in HEAP tables when doing insert + delete + insert + scan the table.
- Fixed bugs on Alpha with REPLACE() [800] and LOAD DATA INFILE.
- Added interactive_timeout [414] variable to mysqld.
- Changed the argument to mysql_data_seek() from ulong to ulonglong.

C.3.55 Changes in Release 3.23.6 (15 December 1999)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added -O lower_case_table_names={0|1} option to mysqld to allow users to force table names to lowercase.
- Added SELECT ... INTO DUMPFILE.
- Added --ansi [384] option to mysqld to make some functions standard SQL compatible.
- Temporary table names now start with #sql.
- Added quoting of identifiers with ` (" in --ansi [384] mode).
- Changed to use snprintf() when printing floats to avoid some buffer overflows on FreeBSD.
- Made FLOOR() [820] overflow safe on FreeBSD.
- Added --quote-names [298] option to mysqldump.
- Fixed bug that one could make a part of a PRIMARY KEY NOT NULL.
- Fixed encrypt() to be thread-safe and not reuse buffer.
- Added mysql_odbc_escape_string() function to support big5 characters in MyODBC.
- Rewrote the storage engine to use classes. This introduces a lot of new code, but make table handling faster and better.
- Added patch by Sasha for user-defined variables.

- Changed that `FLOAT` and `DOUBLE` (without any length modifiers) no longer are fixed decimal point numbers.
- Changed the meaning of `FLOAT(X)`: Now this is the same as `FLOAT` if *X* <= 24 and a `DOUBLE` if 24 < *X* <= 53.
- `DECIMAL(X)` is now an alias for `DECIMAL(X,0)` and `DECIMAL` is now an alias for `DECIMAL(10,0)`. The same goes for `NUMERIC`.
- Added option `ROW_FORMAT={DEFAULT | DYNAMIC | FIXED | COMPRESSED}` to `CREATE_TABLE`.
- `DELETE FROM tbl_name` didn't work on temporary tables.
- Changed function `CHAR_LENGTH()` [794] to be multi-byte character safe.
- Added function `ORD(string)` [799].

C.3.56 Changes in Release 3.23.5 (20 October 1999)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed some Y2K problems in the new date handling in 3.23.
- Fixed problem with `SELECT DISTINCT ... ORDER BY RAND()`.
- Added patches by Sergei A. Golubchik for text searching on the `MyISAM` level.
- Fixed cache overflow problem when using full joins without keys.
- Fixed some configure issues.
- Some small changes to make parsing faster.
- Adding a column after the last field with `ALTER TABLE` didn't work.
- Fixed problem when using an `AUTO_INCREMENT` column in two keys
- With `MyISAM`, you now can have an `AUTO_INCREMENT` column as a key sub part: `CREATE TABLE foo (a INT NOT NULL AUTO_INCREMENT, b CHAR(5), PRIMARY KEY (b,a))`
- Fixed bug in `MyISAM` with packed char keys that could be `NULL`.
- `AS` on field name with `CREATE TABLE tbl_name SELECT ...` didn't work.
- Permit use of `NATIONAL` and `NCHAR` when defining character columns. This is the same as not using `BINARY`.
- Do not allow `NULL` columns in a `PRIMARY KEY` (only in `UNIQUE` keys).
- Clear `LAST_INSERT_ID()` [874] if one uses this in ODBC: `WHERE auto_increment_column IS NULL`. This seems to fix some problems with Access.
- `SET sql_auto_is_null = {0|1}` now turns on/off the handling of searching for the last inserted row with `WHERE auto_increment_column IS NULL`.
- Added new variable `concurrency` to `mysqld` for Solaris.

- Added `--relative` [281] option to `mysqladmin` to make `extended-status` more useful to monitor changes.
- Fixed bug when using `COUNT(DISTINCT ...)` [882] on an empty table.
- Added support for the Chinese character set GBK.
- Fixed problem with `LOAD DATA INFILE` and `BLOB` columns.
- Added bit operator ~ [863] (negation).
- Fixed problem with user-defined functions.

C.3.57 Changes in Release 3.23.4 (28 September 1999)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Inserting a `DATETIME` into a `TIME` column no longer try to store 'days' in it.
- Fixed problem with storage of float/double on little endian machines. (This affected `SUM()` [884].)
- Added connect timeout on TCP/IP connections.
- Fixed problem with `LIKE "%"` [804] on an index that may have `NULL` values.
- `REVOKE ALL PRIVILEGES` didn't revoke all privileges.
- Permit creation of temporary tables with same name as the original table.
- When granting an account a `GRANT` option for a database, the account couldn't grant privileges to other users.
- New statement: `SHOW GRANTS FOR user` (by Sinisa).
- New `date_add` syntax: `date/datetime + INTERVAL # interval_type`. By Joshua Chamas.
- Fixed privilege check for `LOAD DATA REPLACE`.
- Automatic fixing of broken include files on Solaris 2.7
- Some configure issues to fix problems with big file system detection.
- `REGEXP` is now case-insensitive if you use nonbinary strings.

C.3.58 Changes in Release 3.23.3 (13 September 1999)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Added patches for MIT-pthreads on NetBSD.
- Fixed range bug in `MyISAM`.

- `ASC` is now the default again for `ORDER BY`.
- Added `LIMIT` to `UPDATE`.
- Added `mysql_change_user()` function to the MySQL C API.
- Added character set to `SHOW VARIABLES`.
- Added support of `--[whitespace]` comments.
- Permit `INSERT INTO tbl_name VALUES ()`, that is, you may now specify an empty value list to insert a row in which each column is set to its default value.
- Changed `SUBSTRING(text FROM pos)` [802] to conform to standard SQL. (Before this construct returned the rightmost `pos` characters.)
- `SUM()` [884] with `GROUP BY` returned 0 on some systems.
- Changed output for `SHOW TABLE STATUS`.
- Added `DELAY_KEY_WRITE` option to `CREATE TABLE`.
- Permit `AUTO_INCREMENT` on any key part.
- Fixed problem with `YEAR(NOW())` [844] and `YEAR(CURDATE())` [844].
- Added `CASE` [790] construct.
- New `COALESCE()` [784] function.

C.3.59 Changes in Release 3.23.2 (09 August 1999)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed range optimizer bug: `SELECT * FROM tbl_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const)`. The bug was that some rows could be duplicated in the result.
- Running `myisamchk` without `-a` updated the index distribution incorrectly.
- `SET sql_low_priority_updates = 1` was causing a parse error.
- You can now update index columns that are used in the `WHERE` clause. `UPDATE tbl_name SET KEY=KEY+1 WHERE KEY > 100`
- Date handling should now be a bit faster.
- Added handling of fuzzy dates (dates where day or month is 0), such as `'1999-01-00'`.
- Fixed optimization of `SELECT ... WHERE key_part1=const1 AND key_part_2=const2 AND key_part1=const4 AND key_part2=const4`; indextype should be `range` [568] instead of `ref` [567].
- Fixed `egcs` 1.1.2 optimizer bug (when using `BLOB` values) on Linux Alpha.
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`.

- MyISAM tables now allow keys on NULL and BLOB/TEXT columns.
- The following join is now much faster: SELECT ... FROM t1 LEFT JOIN t2 ON ... WHERE t2.*not_null_column* IS NULL.
- ORDER BY and GROUP BY can be done on functions.
- Changed handling of 'const_item' to allow handling of ORDER BY RAND().
- Indexes are now used for WHERE *key_column = function*.
- Indexes are now used for WHERE *key_column = col_name* even if the columns are not identically packed.
- Indexes are now used for WHERE *col_name* IS NULL.
- Changed heap tables to be stored in low_byte_first order (to make it easy to convert to MyISAM tables)
- Automatic change of HEAP temporary tables to MyISAM tables in case of “table is full” errors.
- Added --init-file=*file_name* [387] option to mysqld.
- Added COUNT(DISTINCT *value*, [*value*, ...]) [882].
- CREATE TEMPORARY TABLE now creates a temporary table, in its own namespace, that is automatically deleted if connection is dropped.
- New reserved words (required for CASE [790]): CASE, THEN, WHEN, ELSE and END.
- New functions EXPORT_SET() [795] and MD5() [868].
- Support for the GB2312 Chinese character set.

C.3.60 Changes in Release 3.23.1 (08 July 1999)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- Fixed some compilation problems.

C.3.61 Changes in Release 3.23.0 (05 July 1999: Alpha)

End of Product Lifecycle. Active development and support for MySQL Database Server versions 3.23, 4.0, and 4.1 has ended. For details, see http://www.mysql.com/about/legal/lifecycle/#calendar. Please consider upgrading to a recent version. Further updates to the content of this manual will be minimal. All formats of this manual will continue to be available until 31 Dec 2010.

- A new storage engine library (MyISAM) with a lot of new features. See Section 13.1, “The MyISAM Storage Engine”.
- You can create in-memory HEAP tables which are extremely fast for lookups.
- Support for big files (63-bit) on OSs that support big files.
- New function LOAD_FILE(filename) [798] to get the contents of a file as a string value.

- New `<=>` operator that acts as `=` but returns TRUE if both arguments are `NULL`. This is useful for comparing changes between tables.
- Added the ODBC 3.0 `EXTRACT(interval FROM datetime)` [834] function.
- Columns defined as `FLOAT(X)` are not rounded on storage and may be in scientific notation (1.0 E+10) when retrieved.
- `REPLACE` is now faster than before.
- Changed `LIKE` [804] character comparison to behave as `=` [782]; This means that `'e' LIKE 'é'` is now true. (If the line doesn't display correctly, the latter 'e' is a French 'e' with an acute accent above.)
- `SHOW TABLE STATUS` returns a lot of information about the tables.
- Added `LIKE` [804] to the `SHOW STATUS` statement.
- Added `Privileges` column to `SHOW COLUMNS`.
- Added `Packed` and `Comment` columns to `SHOW INDEX`.
- Added comments to tables (with `CREATE TABLE ... COMMENT 'xxx'`).
- Added `UNIQUE`, as in `CREATE TABLE tbl_name (col INT NOT NULL UNIQUE)`
- New create syntax: `CREATE TABLE tbl_name SELECT ...`
- New create syntax: `CREATE TABLE IF NOT EXISTS ...`
- Permit creation of `CHAR(0)` columns.
- `DATE_FORMAT()` [832] now requires "`%`" before any format character.
- `DELAYED` is now a reserved word (sorry about that :( ).
- An example procedure is added: `analyse`, file: `sql_analyse.c`. This describes the data in your query. Try the following:

```
SELECT ... FROM ...
WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

  This procedure is extremely useful when you want to check the data in your table!
- `BINARY` cast to force a string to be compared in case-sensitive fashion.
- Added `--skip-show-database` [394] option to `mysqld`.
- Check whether a row has changed in an `UPDATE` now also works with `BLOB`/`TEXT` columns.
- Added the `INNER` join syntax. Note that this change makes `INNER` a reserved word!
- Added support for netmasks to the host name in the MySQL grant tables. You can specify a netmask using the `IP/NETMASK` syntax.
- If you compare a `NOT NULL DATE/DATETIME` column with `IS NULL` [783], this is changed to a compare against `0` to satisfy some ODBC applications. (By `<shreeve@uci.edu>`.)
- `NULL IN (...)` now returns `NULL` instead of `0`. This ensures that `null_column NOT IN (...)` doesn't match `NULL` values.

- Fix storage of floating-point values in TIME columns.
- Changed parsing of TIME strings to be more strict. Now the fractional second part is detected (and currently skipped). The following formats are supported:
  - [[DAYS] [H]H:]MM:]SS[.fraction]
  - [[[[[H]H]H]H]MM]SS[.fraction]
- Detect (and ignore) fractional second part from DATETIME.
- Added the LOW_PRIORITY attribute to LOAD DATA INFILE.
- The default index name now uses the same case as the column name on which the index name is based.
- Changed default number of connections to 100.
- Use bigger buffers when using LOAD DATA INFILE.
- DECIMAL(x,y) now works according to standard SQL.
- Added aggregate user-defined functions. Thanks to Andreas F. Bobak (<bobak@relog.ch>) for this!
- LAST_INSERT_ID() [874] is now updated for INSERT INTO ... SELECT.
- Some small changes to the join table optimizer to make some joins faster.
- SELECT DISTINCT is much faster; it uses the new UNIQUE functionality in MyISAM. One difference compared to MySQL 3.22 is that the output of DISTINCT is no longer sorted.
- All C client API macros are now functions to make shared libraries more reliable. Because of this, you can no longer call mysql_num_fields() on a MYSQL object, you must use mysql_field_count() instead.
- Added use of LIBWRAP; patch by Henning P. Schmiedehausen.
- Do not allow AUTO_INCREMENT for other than numeric columns.
- Using AUTO_INCREMENT now automatically makes the column NOT NULL.
- Show NULL as the default value for AUTO_INCREMENT columns.
- Added SQL_BIG_RESULT; SQL_SMALL_RESULT is now default.
- Added a shared library RPM. This enhancement was contributed by David Fox (<dsfox@cogsci.ucsd.edu>).
- Added --enable-large-files and --disable-large-files options to configure. See configure.in for some systems where this is automatically turned off because of broken implementations.
- Upgraded readline to 4.0.
- New CREATE TABLE options: PACK_KEYS and CHECKSUM.
- Added the --default-table-type [386] option to mysqld.

C.4 Changes in InnoDB

Starting from 4.0.22 and 4.1.5, all InnoDB changes are included in the MySQL Change History, and this manual section is no longer separately maintained.

C.4.1 Changes in MySQL/InnoDB-4.0.21, September 10, 2004

Functionality added or changed:

- Renamed the `innodb.status.<pid>` files (created in the data directory) to `innodb_status.<pid>`. This avoids problems on file systems that do not allow multiple periods in file names.
- Added `innodb-status-file` [1066] option to `mysqld` to control whether output from `SHOW INNODB STATUS` is written to a `innodb_status.<pid>` file in the data directory. By default, the file is not created. To create it, start `mysqld` with the `--innodb-status-file=1` [1066] option.
- Changes for NetWare to exit InnoDB gracefully on NetWare even in a case of an assertion failure, instead of intentionally crashing the `` `mysqld' `` server process.

Bugs fixed:

- Fixed a bug in `ON DELETE CASCADE` and `ON UPDATE CASCADE` foreign key constraints: long chains of cascaded operations would cause a stack overflow and crash the server. Cascaded operations are now limited to 15 levels. (Bug #4446)
- Fixed a possible bug in `LOCK TABLES` introduced in MySQL/InnoDB-4.0.19: The count of tables explicitly locked by a transaction was incremented only after the locks were granted, but decremented when the lock structures were destroyed.
- Fixed a bug in `UNLOCK TABLES` in `autocommit = 0` [406] mode, introduced in MySQL/InnoDB-4.0.19: The memory allocated for some locks acquired by the transaction could be deallocated before those locks were released. The bug can lead to crashes and memory corruption of the buffer pool when the transaction acquires a large number of locks (table locks or row-level locks).
- Increment the InnoDB watchdog timeout during `CHECK TABLE`. A long-running `CHECK TABLE` would cause InnoDB to complain about a 'long semaphore wait', and crash the server, if a query had to wait more than 600 seconds behind that `CHECK TABLE` operation. (Bug #2694)
- If you configure `innodb_additional_mem_pool_size` [1066] so small that InnoDB memory allocation spills over from it, then every 4 billionth spill may cause memory corruption. A symptom is a printout like the one following in the `.err` log. The workaround is to make `innodb_additional_mem_pool_size` [1066] big enough to hold all memory allocation. Use `SHOW INNODB STATUS` to determine that there is plenty of free space available in the additional mem pool, and the total allocated memory stays rather constant.

```
InnoDB: Error: Mem area size is 0. Possibly a memory overrun of the
InnoDB: previous allocated area!
InnoDB: Apparent memory corruption: mem dump  len 500; hex
```

- The special meaning of the table names `innodb_monitor`, `innodb_lock_monitor`, `innodb_tablespace_monitor`, `innodb_table_monitor`, and `innodb_validate` in `CREATE TABLE` and `DROP TABLE` statements was accidentally removed in MySQL/InnoDB-4.0.19. The diagnostic functions attached to these special table names (see Section 13.2.14.2, “`SHOW ENGINE INNODB STATUS` and the `InnoDB` Monitors”) are accessible again in MySQL/InnoDB-4.0.21.
- When the private SQL parser of InnoDB was modified in MySQL/InnoDB-4.0.19 to allow the use of the apostrophe (“`'`”) in table and column names, the fix relied on a previously unused function `mem_realloc()`, whose implementation was incorrect. As a result, InnoDB can incorrectly parse

column and table names as the empty string. The InnoDB `realloc()` implementation has been corrected in MySQL/InnoDB-4.0.21.

- Fixed a glitch introduced in 4.0.18 and 4.1.2: in `SHOW TABLE STATUS` InnoDB systematically overestimated the row count by 1 if the table fit on a single 16 kB data page.
- InnoDB created temporary files with the C library function `tmpfile()`. On Windows, the files would be created in the root directory of the current file system. To correct this behavior, the invocations of `tmpfile()` were replaced with code that uses the function `create_temp_file()` in the MySQL portability layer. (Bug #3998)
- If `ALTER TABLE ... DROP FOREIGN KEY ...` fails because of a wrong constraint name, return a table handler error number 150 instead of 152.
- If there was little file I/O in InnoDB, but the insert buffer was used, it could happen that 'Pending normal aio reads' was bigger than 0, but the I/O handler thread did not get waken up in 600 seconds. This resulted in a hang, and crashing of InnoDB.
- If we `RENAME`d a table, InnoDB forgot to load the `FOREIGN KEY` constraints that reference the new table name, and forgot to check that they are compatible with the table.

C.4.2 Changes in MySQL/InnoDB-4.1.4, August 31, 2004

Functionality added or changed:

- 

  Important

  Made internal representation of `TIMESTAMP` values in `InnoDB` in 4.1 to be the same as in 4.0. This difference resulted in incorrect datetime values in `TIMESTAMP` columns in `InnoDB` tables after an upgrade from 4.0 to 4.1. (Bug #4492) **Warning: extra steps during upgrade required!** This means that if you are upgrading from 4.1.x, where x <= 3, to 4.1.4 you should use `mysqldump` for saving and then restoring your `InnoDB` tables with `TIMESTAMP` columns. No conversion is needed if you upgrade from 3.23 or 4.0 to 4.1.4 or later.
- Added a new startup option `innodb_locks_unsafe_for_binlog` [1070]. This option forces `InnoDB` not to use next-key locking in searches and index scans.
- Added `innodb-status-file` [1066] option to `mysqld` to control whether output from `SHOW INNODB STATUS` is written to a `innodb_status.<pid>` file in the data directory. By default, the file is not created. To create it, start `mysqld` with the `--innodb-status-file=1` [1066] option.
- Changes for NetWare to exit InnoDB gracefully on NetWare even in a case of an assertion failure, instead of intentionally crashing the `mysqld` server process.
- “Gap” type row locks without the `LOCK_INSERT_INTENTION` flag do not need to wait for anything. This is because different users can have conflicting lock types on gaps. This change reduces unnecessary deadlocks.

Bugs fixed:

- Fixed a bug in `ON DELETE CASCADE` and `ON UPDATE CASCADE` foreign key constraints: long chains of cascaded operations would cause a stack overflow and crash the server. Cascaded operations are now limited to 15 levels. (Bug #4446)
- Increment the InnoDB watchdog timeout during `CHECK TABLE`. (Bug #2694)

- If you configure `innodb_additional_mem_pool_size` [1066] so small that InnoDB memory allocation spills over from it, then every 4 billionth spill may cause memory corruption. A symptom is a printout like the one following in the `.err` log.

```
InnoDB: Error: Mem area size is 0. Possibly a memory overrun of the
InnoDB: previous allocated area!
InnoDB: Apparent memory corruption: mem dump  len 500; hex
```

- Fixed a glitch introduced in 4.0.18 and 4.1.2: in `SHOW TABLE STATUS` InnoDB systematically overestimated the row count by 1 if the table fit on a single 16 kB data page.
- InnoDB created temporary files with the C library function `tmpfile()`. On Windows, the files would be created in the root directory of the current file system. To correct this behavior, the invocations of `tmpfile()` were replaced with code that uses the function `create_temp_file()` in the MySQL portability layer. (Bug #3998)
- If we `RENAME`d a table, InnoDB forgot to load the foreign key constraints that reference the new table name, and forgot to check that they are compatible with the table.
- If there was little file I/O in InnoDB, but the insert buffer was used, it could happen that 'Pending normal aio reads' was bigger than 0, but the I/O handler thread did not get waken up in 600 seconds. This resulted in a hang, and an intentional crashing of `mysqld`.

C.4.3 Changes in MySQL/InnoDB-4.1.3, June 28, 2004

Functionality added or changed:

- 

  Important

  Starting from MySQL 4.1.3, `InnoDB` uses the same character set comparison functions as MySQL for non-`latin1_swedish_ci` character strings that are not `BINARY`. This changes the sorting order of space and characters < ASCII(32) in those character sets. For `latin1_swedish_ci` character strings and `BINARY` strings, `InnoDB` uses its own pad-spaces-at-end comparison method, which stays unchanged. If you have an `InnoDB` table created with MySQL 4.1.2 or earlier, with an index on a non-`latin1` character set (in the case of 4.1.0 and 4.1.1 with any character set) `CHAR`/`VARCHAR`/or `TEXT` column that is not `BINARY` but may contain characters < ASCII(32), then you should do `ALTER TABLE` or `OPTIMIZE TABLE` on it to **regenerate the index, after upgrading to MySQL 4.1.3 or later**.

- `OPTIMIZE TABLE` for `InnoDB` tables is now mapped to `ALTER TABLE` rather than to `ANALYZE TABLE`.
- Added an interface for storing the binlog offset in the InnoDB log and flushing the log.

Bugs fixed:

- The **critical bug in 4.1.2** (crash recovery skipping all `.ibd` files if you specify `innodb_file_per_table` [1068] on Unix) has been fixed. The bug was a combination of two bugs. Crash recovery ignored the files, because the attempt to lock them in the wrong mode failed. From now on, locks are only obtained for regular files opened in read/write mode, and crash recovery stops if an `.ibd` file for a table exists in a database directory but is unaccessible.
- Do not remember the original `select_lock_type` inside `LOCK TABLES`. (Bug #4047)
- The special meaning of the table names `innodb_monitor`, `innodb_lock_monitor`, `innodb_tablespace_monitor`, `innodb_table_monitor`, and `innodb_validate` in `CREATE`

`TABLE` and `DROP TABLE` statements was accidentally removed in MySQL/InnoDB-4.1.2. The diagnostic functions attached to these special table names (see Section 13.2.14.2, "`SHOW ENGINE INNODB STATUS` and the `InnoDB` Monitors") are accessible again in MySQL/InnoDB-4.1.3.

- When the private SQL parser of InnoDB was modified in MySQL/InnoDB-4.0.19 to allow the use of the apostrophe ("`'`") in table and column names, the fix relied on a previously unused function `mem_realloc()`, whose implementation was incorrect. As a result, InnoDB can incorrectly parse column and table names as the empty string. The InnoDB `realloc()` implementation has been corrected in MySQL/InnoDB-4.1.3.
- In a clean-up of MySQL/InnoDB-4.1.2, the code for invalidating the query cache was broken. Now the query cache should be correctly invalidated for tables affected by `ON UPDATE CASCADE` or `ON DELETE CASCADE` constraints.
- Fixed a bug: in `LIKE 'abc%'` [804], the `'%'` did not match the empty string if the character set was not `latin1_swedish_ci`. This bug was fixed by changing the sorting order in these character sets. See the above note about data conversion in 4.1.3.

C.4.4 Changes in MySQL/InnoDB-4.1.2, May 30, 2004

Note

CRITICAL BUG in 4.1.2 if you specify `innodb_file_per_table` [1068] in `my.cnf` on Unix. In crash recovery InnoDB skips the crash recovery for all `.ibd` files and those tables become CORRUPT! The symptom is a message `Unable to lock ...ibd with lock 1, error: 9: fcntl: Bad file descriptor` in the `.err` log in crash recovery.

Functionality added or changed:

- Support multiple character sets. Note that tables created in other collations than `latin1_swedish_ci` cannot be accessed in MySQL/InnoDB 4.0.
- Automatically create a suitable index on a `FOREIGN KEY`, if the user does not create one. Removes most of the cases of `Error 1005 (errno 150)` in table creation.
- Do not assert in `log0log.c`, line 856 if `ib_logfiles` are too small for `innodb_thread_concurrency` [1072]. Instead, print instructions how to adjust `my.cnf` and call `exit(1)`.
- If MySQL tries to `SELECT` from an InnoDB table without setting any table locks, print a descriptive error message and assert; some subquery bugs were of this type.
- Permit a key-part length in InnoDB to be up to 3,500 bytes; this is needed so that you can create an index on a column with 255 UTF-8 characters.
- All new features from InnoDB-4.0.17, InnoDB-4.0.18, InnoDB-4.0.19 and InnoDB-4.0.20.

Bugs fixed:

- If you configure `innodb_additional_mem_pool_size` [1066] so small that InnoDB memory allocation spills over from it, then every 4 billionth spill may cause memory corruption. A symptom is a printout like the one following in the `.err` log.

```
InnoDB: Error: Mem area size is 0. Possibly a memory overrun of the
InnoDB: previous allocated area!
InnoDB: Apparent memory corruption: mem dump  len 500; hex
```

- Improved portability to 64-bit platforms, especially Win64.
- Fixed an assertion failure when a purge of a table was not possible because of missing `.ibd` file.
- Fixed a bug: do not retrieve all columns in a table if we only need the 'ref' of the row (usually, the `PRIMARY KEY`) to calculate an `ORDER BY`. (Bug #1942)
- On Unix-like systems, obtain an exclusive advisory lock on InnoDB files, to prevent corruption when multiple instances of MySQL are running on the same set of data files. The Windows version of InnoDB currently takes a mandatory lock on the files. (Bug #3608)
- Added a missing space to the output format of `SHOW INNODB STATUS`; reported by Jocelyn Fournier.
- All bugfixes from InnoDB-4.0.17, InnoDB-4.0.18, InnoDB-4.0.19 and InnoDB-4.0.20.

C.4.5 Changes in MySQL/InnoDB-4.0.20, May 18, 2004

Bugs fixed:

- Apostrophe characters now are recognized by the internal `InnoDB` parser and can be used within quoted table and column identifiers in `FOREIGN KEY` clauses.
- Make `LOCK TABLE` aware of `InnoDB` row-level locks and `InnoDB` aware of locks set with `LOCK TABLE`. (Bug #3299)
- Fixed race conditions in `SHOW INNODB STATUS`. (Bug #3596)

C.4.6 Changes in MySQL/InnoDB-4.0.19, May 4, 2004

Functionality added or changed:

- Better error message when the server has to crash because the buffer pool is exhausted by the lock table or the adaptive hash index.
- Print always the count of pending `pread()` and `pwrite()` calls if there is a long semaphore wait.
- Improve space utilization when rows of 1,500 to 8,000 bytes are inserted in the order of the primary key.
- Remove potential buffer overflow errors by sending diagnostic output to stderr or files instead of stdout or fixed-size memory buffers. As a side effect, the output of `SHOW INNODB STATUS` is written to a file `<datadir>/innodb.status.<pid>` every 15 seconds.

Bugs fixed:

- Fixed a bug: `DROP DATABASE` did not work if `FOREIGN KEY` references were defined within the database. (Bug #3058)
- Remove unnecessary files, functions and variables. Many of these were needed in the standalone version of InnoDB. Remove debug functions and variables from nondebug build.
- Add diagnostic code to analyze an assertion failure in ha_innodb.cc on line 2020 reported by a user. (Bug #2903)
- Fixed a bug: in a `FOREIGN KEY`, `ON UPDATE CASCADE` was not triggered if the update changed a string to another value identical in alphabetic ordering, for example, “`abc`” -> “`aBc`”.
- Protect the reading of the latest foreign key error explanation buffer with a mutex; in theory, a race condition could cause `SHOW INNODB STATUS` print garbage characters after the error info.

- Fixed a bug: The row count and key cardinality estimate was grossly too small if each clustered index page only contained one record.
- Parse `CONSTRAINT FOREIGN KEY` correctly. (Bug #3332)
- Fixed a memory corruption bug on Windows. The bug is present in all InnoDB versions in Windows, but it depends on how the linker places a static array in `srv0srv.c`, whether the bug shows itself. 4 bytes were overwritten with a pointer to a statically allocated string '`get windows aio return value`'.
- Fix a glitch reported by Philippe Lewicki on the general mailing list: do not print a warning to the `.err` log if `read_key` fails with a lock wait timeout error 146.
- Permit quotation marks to be embedded in strings in the private SQL parser of InnoDB, so that "`'`" can be used in InnoDB table and column names. Display quotation marks within identifiers properly.
- Debugging: Permit `UNIV_SYNC_DEBUG` to be disabled while `UNIV_DEBUG` is enabled.
- Debugging: Handle magic numbers in a more consistent way.

C.4.7 Changes in MySQL/InnoDB-4.0.18, February 13, 2004

- Do not allow dropping a table referenced by a `FOREIGN KEY` constraint, unless the user does `SET foreign_key_checks = 0`. The error message here is somewhat misleading "Cannot delete or update a parent row...," and must be changed in a future version 4.1.x.
- Make `InnoDB` to remember the `CONSTRAINT` name given by a user for a `FOREIGN KEY`.
- Change the print format of `FOREIGN KEY` constraints spanning multiple databases to `` `db_name`.`tbl_name` ``. But when parsing them, we must also accept `` `db_name.tbl_name` ``, because that was the output format in < 4.0.18.
- An optimization in locking: If `autocommit = 1` [406], then we do not need to make a plain `SELECT` set shared locks even on the `SERIALIZABLE` [976] isolation level, because we know that the transaction is read only. A read-only transaction can always be performed on the `REPEATABLE READ` [976] level, and that does not endanger the serializability.
- Implement an automatic downgrade from >= 4.1.1 -> 4.0.18 if the user has not created tables in `.ibd` files or used other 4.1.x features. **Consult** the manual section on **multiple tablespaces** carefully if you want to downgrade!
- Fixed a bug: MySQL should not allow `REPLACE` to internally perform an `UPDATE` if the table is referenced by a `FOREIGN KEY`. The MySQL manual states that `REPLACE` must resolve a duplicate-key error semantically with `DELETE` + `INSERT`, and not by an `UPDATE`. In versions < 4.0.18 and < 4.1.2, MySQL could resolve a duplicate key conflict in `REPLACE` by doing an `UPDATE` on the existing row, and `FOREIGN KEY` checks could behave in a semantically wrong way. (Bug #2418)
- Fixed a bug: generate `FOREIGN KEY` constraint identifiers locally for each table, in the form *`db_name/tbl_name_ibfk_number`*. If the user gives the constraint name explicitly, then remember it. These changes should ensure that foreign key id's in a slave are the same as in the master, and `DROP FOREIGN KEY` does not break replication. (Bug #2167)
- Fixed a bug: allow quoting of identifiers in InnoDB's `FOREIGN KEY` definitions with a backtick (`) and a double quote ("). You can now use also spaces in table and column names, if you quote the identifiers. (Bug #1725, Bug #2424)
- Fixed a bug: `FOREIGN KEY ... ON UPDATE/DELETE NO ACTION` must check the foreign key constraint, not ignore it. Since we do not have deferred constraints in `InnoDB`, this bugfix makes `InnoDB` to check `NO ACTION` constraints immediately, like it checks `RESTRICT` constraints.

- Fixed a bug: InnoDB crashed in RENAME TABLE if *db_name.tbl_name* is shorter than 5 characters. (Bug #2689)
- Fixed a bug: in SHOW TABLE STATUS, InnoDB row count and index cardinality estimates wrapped around at 512 million in 32-bit computers. Note that unless MySQL is compiled with the big_tables [407] option, they still wrap around at 4 billion.
- Fixed a bug: If there was a UNIQUE secondary index, and NULL values in that unique index, then with the IS NULL [783] predicate, InnoDB returned only the first matching row, though there can be many. This bug was introduced in 4.0.16. (Bug #2483)

C.4.8 Changes in MySQL/InnoDB-5.0.0, December 24, 2003

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0! That is because, for earlier versions, InnoDB is not aware of multiple tablespaces.
- InnoDB in 5.0.0 is essentially the same as InnoDB-4.1.1 with the bugfixes of InnoDB-4.0.17 included.

C.4.9 Changes in MySQL/InnoDB-4.0.17, December 17, 2003

- Fixed a bug: If you created a column prefix secondary index and updated it so that the last characters in the column prefix were spaces, InnoDB would assert in row0upd.c, line 713. The same assertion failed if you updated a column in an ordinary secondary index so that the new value was alphabetically equivalent, but had a different length. This could happen, for example, in the UTF8 character set if you updated a letter to its accented or umlaut form.
- Fixed a bug: InnoDB could think that a secondary index record was not locked though it had been updated to an alphabetically equivalent value, for example, 'abc' -> 'aBc'.
- Fixed a bug: If you updated a secondary index column to an alphabetically equivalent value, and rolled back your update, InnoDB failed to restore the field in the secondary index to its original value.
- There are still several outstanding noncritical bugs reported in the MySQL bugs database. Their fixing has been delayed, because resources were allocated to the 4.1.1 release.

C.4.10 Changes in MySQL/InnoDB-4.1.1, December 4, 2003

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, you cannot downgrade to a version lower than 4.1.1 any more! That is because, for earlier versions, InnoDB is not aware of multiple tablespaces.
- Multiple tablespaces now available for InnoDB. You can store each InnoDB type table and its indexes into a separate .ibd file into a MySQL database directory, into the same directory where the .frm file is stored.
- The MySQL query cache now works for InnoDB tables also if autocommit = 0 [406], or the statements are enclosed inside BEGIN ... COMMIT.
- Reduced InnoDB memory consumption by a few megabytes if one sets the buffer pool size < 8MB.
- You can use raw disk partitions also in Windows.

C.4.11 Changes in MySQL/InnoDB-4.0.16, October 22, 2003

- Fixed a bug: in contrary to what was said in the manual, in a locking read InnoDB set two record locks if a unique exact match search condition was used on a multi-column unique key. For a single column unique key it worked right.

- Fixed a bug: If you used the rename trick `#sql... -> rsql...` to recover a temporary table, `InnoDB` asserted in `row_mysql_lock_data_dictionary()`.
- There are several outstanding noncritical bugs reported in the MySQL bugs database. Their fixing has been delayed, because resources are allocated to the upcoming 4.1.1 release.

C.4.12 Changes in MySQL/InnoDB-3.23.58, September 15, 2003

- Fixed a bug: `InnoDB` could make the index page directory corrupt in the first B-tree page splits after `mysqld` startup. A symptom would be an assertion failure in `page0page.c`, in function `page_dir_find_slot()`.
- Fixed a bug: `InnoDB` could in rare cases return an extraneous row if a rollback, purge, and a `SELECT` coincided.
- Fixed a possible hang over the `btr0sea.c` latch if `SELECT` was used inside `LOCK TABLES`.
- Fixed a bug: If a single `DELETE` statement first managed to delete some rows and then failed in a `FOREIGN KEY` error or a `Table is full` error, MySQL did not roll back the whole SQL statement as it should.

C.4.13 Changes in MySQL/InnoDB-4.0.15, September 10, 2003

- Fixed a bug: If you updated a row so that the 8000 byte maximum length (without `BLOB` and `TEXT`) was exceeded, `InnoDB` simply removed the record from the clustered index. In a similar insert, `InnoDB` would leak reserved file space extents, which would only be freed at the next `mysqld` startup.
- Fixed a bug: If you used big `BLOB` values, and your log files were relatively small, `InnoDB` could in a big `BLOB` operation temporarily write over the log produced after the latest checkpoint. If `InnoDB` would crash at that moment, then the crash recovery would fail, because `InnoDB` would not be able to scan the log even up to the latest checkpoint. Starting from this version, `InnoDB` tries to ensure the latest checkpoint is young enough. If that is not possible, `InnoDB` prints a warning to the `.err` log of MySQL and advises you to make the log files bigger.
- Fixed a bug: setting `innodb_fast_shutdown = 0` had no effect.
- Fixed a bug introduced in 4.0.13: If a `CREATE TABLE` ended in a comment, that could cause a memory overrun.
- Fixed a bug: If `InnoDB` printed `Operating system error number .. in a file operation` to the `.err` log in Windows, the error number explanation was wrong. Workaround: See Section 13.2.13.2, “Operating System Error Codes”, about Windows error numbers.
- Fixed a bug: If you created a column prefix `PRIMARY KEY` like in `t(a CHAR(200), PRIMARY KEY (a(10)))` on a fixed-length `CHAR` column, `InnoDB` would crash even in a simple `SELECT`. A `CHECK TABLE` would report the table as corrupt, also in the case where the created key was not `PRIMARY`.

C.4.14 Changes in MySQL/InnoDB-4.0.14, July 22, 2003

- `InnoDB` now supports the `SAVEPOINT` and `ROLLBACK TO SAVEPOINT` SQL statements. For the syntax, see Section 12.3.4, “`SAVEPOINT` and `ROLLBACK TO SAVEPOINT` Syntax”, and Section 12.3.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”.
- You can now create column prefix keys like in `CREATE TABLE t (a BLOB, INDEX (a(10)))`.
- You can also use `O_DIRECT` as the `innodb_flush_method` [1069] on the latest versions of Linux and FreeBSD. Beware of possible bugs in those operating systems, though.

- Fixed the checksum calculation of data pages. Previously most OS file system corruption went unnoticed. Note that if you downgrade from version 4.0.14 or up to a version earlier than 4.0.14, `InnoDB` prints warnings in the first startup:

```
InnoDB: Warning: An inconsistent page in the doublewrite buffer
InnoDB: space id 2552202359 page number 8245, 127'th page in dblwr buf.
```

  but that is not dangerous and can be ignored.

- Modified the buffer pool replacement algorithm so that it tries to flush modified pages if there are no replaceable pages in the last 10% of the LRU list. This can reduce disk I/O if the workload is a mixture of reads and writes.

- The buffer pool checkpoint flush algorithm now tries to flush also close neighbors of the page at the end of the flush list. This can speed up database shutdown, and can also speed up disk writes if `InnoDB` log files are very small compared to the buffer pool size.

- In 4.0.13 we made `SHOW INNODB STATUS` to print detailed info on the latest `UNIQUE KEY` error, but storing that information could slow down `REPLACE` significantly. We no longer store or print the info.

- Fixed a bug: `SET foreign_key_checks = 0` was not replicated properly in the MySQL replication. The fix will not be backported to 3.23.

- Fixed a bug: the parameter `innodb_max_dirty_pages_pct` [1071] forgot to take into account the free pages in the buffer pool. This could lead to excessive flushing even though there were lots of free pages in the buffer pool. Workaround: `SET GLOBAL innodb_max_dirty_pages_pct = 100`.

- Fixed a bug: If there were big index scans then a file read request could starve and `InnoDB` could assert because of a very long semaphore wait.

- Fixed a bug: If `autocommit = 1` [406] then inside `LOCK TABLES` MySQL failed to do the commit after an updating SQL statement if binary logging was not on, and for `SELECT` statements did not commit regardless of binary logging state.

- Fixed a bug: `InnoDB` could make the index page directory corrupt in the first B-tree page splits after a `mysqld` startup. A symptom would be an assertion in page0page.c, in function page_dir_find_slot().

- Fixed a bug: If in a `FOREIGN KEY` with an `UPDATE CASCADE` clause the parent column was of a different internal storage length than the child column, then a cascaded update would make the column length wrong in the child table and corrupt the child table. Because of MySQL's 'silent column specification changes' a fixed-length `CHAR` column can change internally to a `VARCHAR` and cause this error.

- Fixed a bug: If a non-`latin1` character set was used and if in a `FOREIGN KEY` the parent column was of a different internal storage length than the child column, then all inserts to the child table would fail in a foreign key error.

- Fixed a bug: `InnoDB` could complain that it cannot find the clustered index record, or in rare cases return an extraneous row if a rollback, purge, and a `SELECT` coincided.

- Fixed a possible hang over the btr0sea.c latch if `SELECT` was used inside `LOCK TABLES`.

- Fixed a bug: contrary to what the release note of 4.0.13 said, the group commit still did not work if the MySQL binary logging was on.

- Fixed a bug: os_event_wait() did not work properly in Unix, which might have caused starvation in various log operations.

- Fixed a bug: If a single `DELETE` statement first managed to delete some rows and then failed in a `FOREIGN KEY` error or a `Table is full` error, MySQL did not roll back the whole SQL statement as it should, and also wrote the failed statement to the binary log, reporting there a nonzero error_code.
- Fixed a bug: the maximum permitted number of columns in a table is 1000, but `InnoDB` did not check that limit in `CREATE TABLE`, and a subsequent `INSERT` or `SELECT` from that table could cause an assertion.

C.4.15 Changes in MySQL/InnoDB-3.23.57, June 20, 2003

- Changed the default value of `innodb_flush_log_at_trx_commit` [1068] from 0 to 1. If you have not specified it explicitly in your `my.cnf`, and your application runs much slower with this new release, it is because the value 1 causes a log flush to disk at each transaction commit.
- Fixed a bug: `InnoDB` forgot to call pthread_mutex_destroy() when a table was dropped. That could cause memory leakage on FreeBSD and other non-Linux Unixes.
- Fixed a bug: MySQL could erroneously return 'Empty set' if `InnoDB` estimated an index range size to 0 records though the range was not empty; MySQL also failed to do the next-key locking in the case of an empty index range.
- Fixed a bug: `GROUP BY` and `DISTINCT` could treat NULL values inequal.

C.4.16 Changes in MySQL/InnoDB-4.0.13, May 20, 2003

- `InnoDB` now supports `ALTER TABLE DROP FOREIGN KEY`. You have to use `SHOW CREATE TABLE` to find the internally generated foreign key ID when you want to drop a foreign key.
- `SHOW INNODB STATUS` now prints detailed information of the latest detected `FOREIGN KEY` and `UNIQUE KEY` errors. If you do not understand why `InnoDB` gives the error 150 from a `CREATE TABLE`, you can use this statement to study the reason.
- `ANALYZE TABLE` now works also for `InnoDB` type tables. It makes eight random dives to each of the index trees and updates index cardinality estimates accordingly. Note that because these are only estimates, repeated runs of `ANALYZE TABLE` may produce different numbers. MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you may try using `ANALYZE TABLE`.
- `InnoDB` group commit capability now works also when MySQL binary logging is switched on. There have to be > 2 client threads for the group commit to become active.
- Changed the default value of `innodb_flush_log_at_trx_commit` [1068] from 0 to 1. If you have not specified it explicitly in your `my.cnf`, and your application runs much slower with this new release, it is because the value 1 causes a log flush to disk at each transaction commit.
- Added a new global settable MySQL system variable `innodb_max_dirty_pages_pct` [1071]. It is an integer in the range 0 - 100. The default is 90. The main thread in `InnoDB` tries to flush pages from the buffer pool so that at most this many percents are not yet flushed at any time.
- If `innodb_force_recovery=6`, do not let `InnoDB` do repair of corrupt pages based on the doublewrite buffer.
- `InnoDB` startup now happens faster because it does not set the memory in the buffer pool to zero.
- Fixed a bug: The `InnoDB` parser for `FOREIGN KEY` definitions was confused by the keywords 'foreign key' inside MySQL comments.

- Fixed a bug: If you dropped a table to which there was a `FOREIGN KEY` reference, and later created the same table with nonmatching data types, `InnoDB` could assert in `dict0load.c`, in function `dict_load_table()`.
- Fixed a bug: `GROUP BY` and `DISTINCT` could treat `NULL` values as not equal. MySQL also failed to do the next-key locking in the case of an empty index range.
- Fixed a bug: Do not commit the current transaction when a `MyISAM` table is updated; this also makes `CREATE TABLE` not to commit an `InnoDB` transaction, even when binary logging is enabled.
- Fixed a bug: We did not allow `ON DELETE SET NULL` to modify the same table where the delete was made; we can allow it because that cannot produce infinite loops in cascaded operations.
- Fixed a bug: Enable `HANDLER PREV` and `NEXT` also after positioning the cursor with a unique search on the primary key.
- Fixed a bug: If `MIN()` [884] or `MAX()` [884] resulted in a deadlock or a lock wait timeout, MySQL did not return an error, but returned `NULL` as the function value.
- Fixed a bug: `InnoDB` forgot to call `pthread_mutex_destroy()` when a table was dropped. That could cause memory leakage on FreeBSD and other non-Linux Unix systems.

C.4.17 Changes in MySQL/InnoDB-4.1.0, April 3, 2003

- `InnoDB` now supports up to 64GB of buffer pool memory in a Windows 32-bit Intel computer. This is possible because `InnoDB` can use the AWE extension of Windows to address memory over the 4GB limit of a 32-bit process. A new startup variable `innodb_buffer_pool_awe_mem_mb` [1067] enables AWE and sets the size of the buffer pool in megabytes.
- Reduced the size of buffer headers and the lock table. `InnoDB` uses 2% less memory.

C.4.18 Changes in MySQL/InnoDB-3.23.56, March 17, 2003

- Fixed a major bug in `InnoDB` query optimization: queries of type SELECT ... WHERE indexcolumn < x and SELECT ... WHERE indexcolumn > x could cause a table scan even if the selectivity would have been very good.
- Fixed a potential bug if MySQL calls store_lock with TL_IGNORE in the middle of a query.

C.4.19 Changes in MySQL/InnoDB-4.0.12, March 18, 2003

- In crash recovery `InnoDB` now prints the progress in percents of a transaction rollback.
- Fixed a bug/feature: If your application program used mysql_use_result(), and used >= 2 connections to send SQL queries, it could deadlock on the adaptive hash S-latch in btr0sea.c. Now `mysqld` releases the S-latch whenever it passes data from a SELECT to the client.
- Fixed a bug: MySQL could erroneously return 'Empty set' if `InnoDB` estimated an index range size to 0 records though the range was not empty; MySQL also failed to do the next-key locking in the case of an empty index range.

C.4.20 Changes in MySQL/InnoDB-4.0.11, February 25, 2003

- Fixed a bug introduced in 4.0.10: `SELECT ... FROM ... ORDER BY ... DESC` could hang in an infinite loop.
- An outstanding bug: `SET foreign_key_checks = 0` is not replicated properly in the MySQL replication.

C.4.21 Changes in MySQL/InnoDB-4.0.10, February 4, 2003

- In INSERT INTO t1 SELECT ... FROM t2 WHERE ... MySQL previously set a table level read lock on t2. This lock is now removed.
- Increased SHOW INNODB STATUS maximum printed length to 200KB.
- Fixed a major bug in `InnoDB` query optimization: queries of type SELECT ... WHERE indexcolumn < x and SELECT ... WHERE indexcolumn > x could cause a table scan even if the selectivity would have been very good.
- Fixed a bug: purge could cause a hang in a BLOB table where the primary key index tree was of height 1. Symptom: semaphore waits caused by an X-latch set in btr_free_externally_stored_field().
- Fixed a bug: using `InnoDB` HANDLER commands on a fresh handle crashed `mysqld` in `ha_innobase::change_active_index()`.
- Fixed a bug: If MySQL estimated a query in the middle of a SELECT statement, `InnoDB` could hang on the adaptive hash index latch in btr0sea.c.
- Fixed a bug: `InnoDB` could report table corruption and assert in page_dir_find_owner_slot() if an adaptive hash index search coincided with purge or an insert.
- Fixed a bug: some file system snapshot tool in Windows 2000 could cause an `InnoDB` file write to fail with error 33 ERROR_LOCK_VIOLATION. In synchronous writes `InnoDB` now retries the write 100 times at 1 second intervals.
- Fixed a bug: `REPLACE INTO t1 SELECT ...` did not work if t1 has an `AUTO_INCREMENT` column.
- An outstanding bug: `SET foreign_key_checks = 0` is not replicated properly in the MySQL replication.

C.4.22 Changes in MySQL/InnoDB-3.23.55, January 24, 2003

- In INSERT INTO t1 SELECT ... FROM t2 WHERE ... MySQL previously set a table level read lock on t2. This lock is now removed.
- Fixed a bug: If the combined size of `InnoDB` log files was >= 2GB in a 32-bit computer, `InnoDB` would write log in a wrong position. That could make crash recovery and `InnoDB Hot Backup` to fail in log scan.
- Fixed a bug: index cursor restoration could theoretically fail.
- Fixed a bug: an assertion in btr0sea.c, in function btr_search_info_update_slow could theoretically fail in a race of 3 threads.
- Fixed a bug: purge could cause a hang in a BLOB table where the primary key index tree was of height 1. Symptom: semaphore waits caused by an X-latch set in btr_free_externally_stored_field().
- Fixed a bug: If MySQL estimated a query in the middle of a SELECT statement, `InnoDB` could hang on the adaptive hash index latch in btr0sea.c.
- Fixed a bug: `InnoDB` could report table corruption and assert in page_dir_find_owner_slot() if an adaptive hash index search coincided with purge or an insert.
- Fixed a bug: some file system snapshot tool in Windows 2000 could cause an `InnoDB` file write to fail with error 33 ERROR_LOCK_VIOLATION. In synchronous writes `InnoDB` now retries the write 100 times at 1 second intervals.

- An outstanding bug: `SET foreign_key_checks = 0` is not replicated properly in the MySQL replication. The fix appears in 4.0.11 and probably will not be backported to 3.23.
- Fixed bug in `InnoDB` `page0cur.c` file in function page_cur_search_with_match which caused `InnoDB` to remain on the same page forever. This bug is evident only in tables with more than one page.

C.4.23 Changes in MySQL/InnoDB-4.0.9, January 14, 2003

- Removed the warning message: 'InnoDB: Out of memory in additional memory pool.'
- Fixed a bug: If the combined size of `InnoDB` log files was >= 2GB in a 32-bit computer, `InnoDB` would write log in a wrong position. That could make crash recovery and `InnoDB Hot Backup` to fail.
- Fixed a bug: index cursor restoration could theoretically fail.

C.4.24 Changes in MySQL/InnoDB-4.0.8, January 7, 2003

- `InnoDB` now supports also FOREIGN KEY (...) REFERENCES ...(...) [ON UPDATE CASCADE | ON UPDATE SET NULL | ON UPDATE RESTRICT | ON UPDATE NO ACTION].
- Tables and indexes now reserve 4% less space in the tablespace. Also existing tables reserve less space. By upgrading to 4.0.8 you should see more free space in "InnoDB free" in SHOW TABLE STATUS.
- Fixed bugs: updating the PRIMARY KEY of a row would generate a foreign key error on all FOREIGN KEYs which referenced secondary keys of the row to be updated. Also, if a referencing FOREIGN KEY constraint only referenced the first columns in an index, and there were more columns in that index, updating the additional columns generated a foreign key error.
- Fixed a bug: If an index contains some column twice, and that column is updated, the table becomes corrupt. From now on `InnoDB` prevents creation of such indexes.
- Fixed a bug: removed superfluous error 149 and 150 printouts from the .err log when a locking SELECT caused a deadlock or a lock wait timeout.
- Fixed a bug: an assertion in btr0sea.c, in function btr_search_info_update_slow could theoretically fail in a race of 3 threads.
- Fixed a bug: one could not switch a session transaction isolation level back to REPEATABLE READ after setting it to something else.

C.4.25 Changes in MySQL/InnoDB-4.0.7, December 26, 2002

- `InnoDB` in 4.0.7 is essentially the same as in 4.0.6.

C.4.26 Changes in MySQL/InnoDB-4.0.6, December 19, 2002

- Since innodb_log_arch_dir has no relevance under MySQL, there is no need to specify it any more in my.cnf.
- `LOAD DATA INFILE` in `autocommit = 1` [406] mode no longer does implicit commits for each 1MB of written binary log.
- Fixed a bug introduced in 4.0.4: LOCK TABLES ... READ LOCAL should not set row locks on the rows read. This caused deadlocks and lock wait timeouts in mysqldump.
- Fixed two bugs introduced in 4.0.4: in AUTO_INCREMENT, REPLACE could cause the counter to be left 1 too low. A deadlock or a lock wait timeout could cause the same problem.

- Fixed a bug: TRUNCATE on a TEMPORARY table crashed InnoDB.
- Fixed a bug introduced in 4.0.5: If binary logging was not switched on, INSERT INTO ... SELECT ... or CREATE TABLE ... SELECT ... could cause InnoDB to hang on a semaphore created in btr0sea.c, line 128. Workaround: switch binary logging on.
- Fixed a bug: in replication issuing STOP SLAVE in the middle of a multiple-statement transaction could cause that START SLAVE would only perform a part of the transaction. A similar error could occur if the slave crashed and was restarted.

C.4.27 Changes in MySQL/InnoDB-3.23.54, December 12, 2002

- Fixed a bug: the InnoDB range estimator greatly exaggerated the size of a short index range if the paths to the endpoints of the range in the index tree happened to branch in the root. This could cause unnecessary table scans in SQL queries.
- Fixed a bug: ORDER BY could fail if you had not created a primary key to a table, but had defined several indexes of which at least one was a UNIQUE index with all its columns declared as NOT NULL.
- Fixed a bug: a lock wait timeout in connection with ON DELETE CASCADE could cause corruption in indexes.
- Fixed a bug: If a SELECT was done with a unique key from a primary index, and the search matched to a delete-marked record, InnoDB could erroneously return the NEXT record.
- Fixed a bug introduced in 3.23.53: LOCK TABLES ... READ LOCAL should not set row locks on the rows read. This caused deadlocks and lock wait timeouts in mysqldump.
- Fixed a bug: If an index contains some column twice, and that column is updated, the table becomes corrupt. From now on InnoDB prevents creation of such indexes.

C.4.28 Changes in MySQL/InnoDB-4.0.5, November 18, 2002

- InnoDB now supports also transaction isolation levels READ COMMITTED and READ UNCOMMITTED. READ COMMITTED more closely emulates Oracle and makes porting of applications from Oracle to MySQL easier.
- Deadlock resolution is now selective: we try to pick as victims transactions with less modified or inserted rows.
- FOREIGN KEY definitions are now aware of the lower_case_table_names setting in my.cnf.
- SHOW CREATE TABLE does not output the database name to a FOREIGN KEY definition if the referred table is in the same database as the table.
- InnoDB does a consistency check to most index pages before writing them to a data file.
- If you set innodb_force_recovery [1069] > 0, InnoDB tries to jump over corrupt index records and pages when doing SELECT * FROM table. This helps in dumping.
- InnoDB now again uses asynchronous unbuffered I/O in Windows 2000 and XP; only unbuffered simulated async I/O in NT, 95/98/ME.
- Fixed a bug: the InnoDB range estimator greatly exaggerated the size of a short index range if the paths to the endpoints of the range in the index tree happened to branch in the root. This could cause unnecessary table scans in SQL queries. The fix is also backported to 3.23.54.
- Fixed a bug present in 3.23.52, 4.0.3, 4.0.4: InnoDB startup could take very long or even crash on some Windows 95/98/ME computers.

- Fixed a bug: the AUTO-INC lock was held to the end of the transaction if it was granted after a lock wait. This could cause unnecessary deadlocks.
- Fixed a bug: If SHOW INNODB STATUS, innodb_monitor, or innodb_lock_monitor had to print several hundred transactions in one report, and the output became truncated, `InnoDB` would hang, printing to the error log many waits for a mutex created at srv0srv.c, line 1621.
- Fixed a bug: SHOW INNODB STATUS on Unix always reported average file read size as 0 bytes.
- Fixed a potential bug in 4.0.4: `InnoDB` now does ORDER BY ... DESC like `MyISAM`.
- Fixed a bug: DROP TABLE could cause crash or a hang if there was a rollback concurrently running on the table. The fix will be backported to 3.23 only if this appears a real problem for users.
- Fixed a bug: ORDER BY could fail if you had not created a primary key to a table, but had defined several indexes of which at least one was a UNIQUE index with all its columns declared as NOT NULL.
- Fixed a bug: a lock wait timeout in connection with ON DELETE CASCADE could cause corruption in indexes.
- Fixed a bug: If a SELECT was done with a unique key from a primary index, and the search matched to a delete-marked record, `InnoDB` could return the NEXT record.
- Outstanding bugs: in 4.0.4 two bugs were introduced to AUTO_INCREMENT. REPLACE can cause the counter to be left 1 too low. A deadlock or a lock wait timeout can cause the same problem. These are fixed in 4.0.6.

C.4.29 Changes in MySQL/InnoDB-3.23.53, October 9, 2002

- We again use unbuffered disk I/O to data files in Windows. Windows XP and Windows 2000 read performance seems to be very poor with normal I/O.
- Tuned range estimator so that index range scans are preferred over full index scans.
- Enable dropping and creating a table even if `innodb_force_recovery` [1069] is set. One can use this to drop a table which would cause a crash in rollback or purge, or if a failed table import causes a runaway rollback in recovery.
- Fixed a bug present in 3.23.52, 4.0.3, 4.0.4: `InnoDB` startup could take very long or even crash on some Windows 95/98/ME computers.
- Fixed a bug: fast shutdown (which is the default) sometimes was slowed down by purge and insert buffer merge.
- Fixed a bug: doing a big SELECT from a table where no rows were visible in a consistent read could cause a very long (> 600 seconds) semaphore wait in btr0cur.c line 310.
- Fixed a bug: the AUTO-INC lock was held to the end of the transaction if it was granted after a lock wait. This could cause unnecessary deadlocks.
- Fixed a bug: If you created a temporary table inside LOCK TABLES, and used that temporary table, that caused an assertion failure in ha_innobase.cc.
- Fixed a bug: If SHOW INNODB STATUS, innodb_monitor, or innodb_lock_monitor had to print several hundred transactions in one report, and the output became truncated, `InnoDB` would hang, printing to the error log many waits for a mutex created at srv0srv.c, line 1621.
- Fixed a bug: SHOW INNODB STATUS on Unix always reported average file read size as 0 bytes.

C.4.30 Changes in MySQL/InnoDB-4.0.4, October 2, 2002

- We again use unbuffered disk I/O in Windows. Windows XP and Windows 2000 read performance seems to be very poor with normal I/O.
- Increased the maximum key length of `InnoDB` tables from 500 to 1024 bytes.
- Increased the table comment field in SHOW TABLE STATUS so that up to 16000 characters of foreign key definitions can be printed there.
- The auto-increment counter is no longer incremented if an insert of a row immediately fails in an error.
- Enable dropping and creating a table even if `innodb_force_recovery` [1069] is set. One can use this to drop a table which would cause a crash in rollback or purge, or if a failed table import causes a runaway rollback in recovery.
- Fixed a bug: Using ORDER BY primarykey DESC in 4.0.3 causes an assertion failure in btr0pcur.c, line 203.
- Fixed a bug: fast shutdown (which is the default) sometimes was slowed down by purge and insert buffer merge.
- Fixed a bug: doing a big SELECT from a table where no rows were visible in a consistent read could cause a very long (> 600 seconds) semaphore wait in btr0cur.c line 310.
- Fixed a bug: If the MySQL query cache was used, it did not get invalidated by a modification done by ON DELETE CASCADE or ...SET NULL.
- Fixed a bug: If you created a temporary table inside LOCK TABLES, and used that temporary table, that caused an assertion failure in ha_innodb.cc.
- Fixed a bug: If you set innodb_flush_log_at_trx_commit to 1, SHOW VARIABLES would show its value as 16 million.

C.4.31 Changes in MySQL/InnoDB-4.0.3, August 28, 2002

- Removed unnecessary deadlocks when inserts have to wait for a locking read, update, or delete to release its next-key lock.
- The MySQL `HANDLER` SQL statements now work also for `InnoDB` type tables. `InnoDB` does the `HANDLER` reads always as consistent reads. `HANDLER` is a direct access path to read individual indexes of tables. In some cases, `HANDLER` can be used as a substitute of server-side cursors.
- Fixed a bug in 4.0.2: even a simple insert could crash the AIX version.
- Fixed a bug: If you used in a table name characters whose code is > 127, in DROP TABLE `InnoDB` could assert on line 155 of pars0sym.c.
- Compilation from source now provides a working version both on HP-UX-11 and HP-UX-10.20. The source of 4.0.2 worked only on 11, and the source of 3.23.52 only on 10.20.
- Fixed a bug: If compiled on 64-bit Solaris, `InnoDB` produced a bus error at startup.

C.4.32 Changes in MySQL/InnoDB-3.23.52, August 16, 2002

- The feature set of 3.23 is frozen from this version on. New features go the 4.0 branch, and only bugfixes are made to the 3.23 branch.

- Many CPU-bound join queries now run faster. On Windows also many other CPU-bound queries run faster.
- A new SQL statement SHOW INNODB STATUS returns the output of the `InnoDB` Monitor to the client. The `InnoDB` Monitor now prints detailed information on the latest detected deadlock.
- `InnoDB` made the SQL query optimizer to avoid too much index-only range scans and choose full table scans instead. This is now fixed.
- `BEGIN` and `COMMIT` are now added in the binary log around transactions. The MySQL replication now respects transaction borders: a user no longer sees half transactions in replication slaves.
- A replication slave now prints in crash recovery the last master binary log position it was able to recover to.
- A new setting `innodb_flush_log_at_trx_commit=2` makes `InnoDB` to write the log to the operating system file cache at each commit. This is almost as fast as the setting innodb_flush_log_at_trx_commit = 0, and the setting 2 also has the nice feature that in a crash where the operating system does not crash, no committed transaction is lost. If the operating system crashes or there is a power outage, then the setting 2 is no safer than the setting 0.
- Added checksum fields to log blocks.
- `SET foreign_key_checks = 0` helps in importing tables in an arbitrary order which does not respect the foreign key rules.
- `SET unique_checks = 0` speeds up table imports into `InnoDB` if you have UNIQUE constraints on secondary indexes. This flag should be used only if you are certain that the input records contain no UNIQUE constraint violations.
- SHOW TABLE STATUS now lists also possible ON DELETE CASCADE or ON DELETE SET NULL in the comment field of the table.
- When CHECK TABLE is run on any `InnoDB` type table, it now checks also the adaptive hash index for all tables.
- If you defined ON DELETE CASCADE or SET NULL and updated the referenced key in the parent row, `InnoDB` deleted or updated the child row. This is now changed to conform to standard SQL: you get the error 'Cannot delete parent row'.
- Improved the auto-increment algorithm: now the first insert or SHOW TABLE STATUS initializes the auto-increment counter for the table. This removes almost all surprising deadlocks caused by SHOW TABLE STATUS.
- Aligned some buffers used in reading and writing to data files. This enables using unbuffered raw devices as data files in Linux.
- Fixed a bug: If you updated the primary key of a table so that only the case of characters changed, that could cause assertion failures, mostly in page0page.ic line 515.
- Fixed a bug: If you delete or update a row referenced in a foreign key constraint and the foreign key check has to wait for a lock, then the check may report an erroneous result. This affects also the ON DELETE... operation.
- Fixed a bug: A deadlock or a lock wait timeout error in `InnoDB` causes `InnoDB` to roll back the whole transaction, but MySQL could still write the earlier SQL statements to the binary log, even though `InnoDB` rolled them back. This could, for example, cause replicated databases to get out-of-sync.

- Fixed a bug: If the database happened to crash in the middle of a commit, then the recovery might leak tablespace pages.
- Fixed a bug: If you specified a non-`latin1` character set in my.cnf, then, in contrary to what is stated in the manual, in a foreign key constraint a string type column had to have the same length specification in the referencing table and the referenced table.
- Fixed a bug: DROP TABLE or DROP DATABASE could fail if there simultaneously was a CREATE TABLE running.
- Fixed a bug: If you configured the buffer pool bigger than 2GB in a 32-bit computer, `InnoDB` would assert in buf0buf.ic line 214.
- Fixed a bug: on 64-bit computers updating rows which contained the SQL NULL in some column could cause the undo log and the ordinary log to become corrupt.
- Fixed a bug: innodb_log_monitor caused a hang if it suppressed lock prints for a page.
- Fixed a bug: in the HP-UX-10.20 version mutexes would leak and cause race conditions and crashes in any part of `InnoDB` code.
- Fixed a bug: If you ran in the `autocommit` [406] mode, executed a SELECT, and immediately after that a RENAME TABLE, then RENAME would fail and MySQL would complain about error 1192.
- Fixed a bug: If compiled on 64-bit Solaris, `InnoDB` produced a bus error at startup.

C.4.33 Changes in MySQL/InnoDB-4.0.2, July 10, 2002

- `InnoDB` is essentially the same as InnoDB-3.23.51.
- If no innodb_data_file_path is specified, `InnoDB` at the database creation now creates a 10MB auto-extending data file ibdata1 to the datadir of MySQL. In 4.0.1 the file was 64MB and not auto-extending.

C.4.34 Changes in MySQL/InnoDB-3.23.51, June 12, 2002

- Fixed a bug: a join could result in a segmentation fault in copying of a BLOB or TEXT column if some of the BLOB or TEXT columns in the table contained SQL NULL values.
- Fixed a bug: If you added self-referential foreign key constraints with ON DELETE CASCADE to tables and a row deletion caused `InnoDB` to attempt the deletion of the same row twice because of a cascading delete, then you got an assertion failure.
- Fixed a bug: If you use MySQL 'user-level locks' and close a connection, then `InnoDB` may assert in ha_innobase.cc, line 302.

C.4.35 Changes in MySQL/InnoDB-3.23.50, April 23, 2002

- `InnoDB` now supports an auto-extending last data file. You do not need to preallocate the whole data file at the database startup.
- Made several changes to facilitate the use of the `InnoDB Hot Backup` tool. It is a separate nonfree tool you can use to take online backups of your database without shutting down the server or setting any locks.
- If you want to run the `InnoDB Hot Backup` tool on an auto-extending data file you have to upgrade it to version ibbackup-0.35.
- The log scan phase in crash recovery now runs much faster.

- Starting from this server version, the hot backup tool truncates unused ends in the backup `InnoDB` data files.
- To allow the hot backup tool to work, on Windows we no longer use unbuffered I/O or native async I/O; instead we use the same simulated async I/O as on Unix.
- You can now define the ON DELETE CASCADE or ON DELETE SET NULL clause on foreign keys.
- FOREIGN KEY constraints now survive ALTER TABLE and CREATE INDEX.
- We suppress the FOREIGN KEY check if any of the column values in the foreign key or referenced key to be checked is the SQL NULL. This is compatible with Oracle, for example.
- SHOW CREATE TABLE now lists also foreign key constraints. Also mysqldump no longer forgets about foreign keys in table definitions.
- You can now add a new foreign key constraint with ALTER TABLE ... ADD CONSTRAINT FOREIGN KEY (...) REFERENCES ... (...).
- FOREIGN KEY definitions now allow backticks around table and column names.
- MySQL command SET TRANSACTION ISOLATION LEVEL ... has now the following effect on `InnoDB` tables: If a transaction is defined as SERIALIZABLE then `InnoDB` conceptually adds LOCK IN SHARE MODE to all consistent reads. If a transaction is defined to have any other isolation level, then `InnoDB` obeys its default locking strategy which is REPEATABLE READ.
- SHOW TABLE STATUS no longer sets an x-lock at the end of an auto-increment index if the auto-increment counter has been initialized. This removes in almost all cases the surprising deadlocks caused by SHOW TABLE STATUS.
- Fixed a bug: in a CREATE TABLE statement the string 'foreign' followed by a nonspace character confused the FOREIGN KEY parser and caused table creation to fail with errno 150.

C.4.36 Changes in MySQL/InnoDB-3.23.49, February 17, 2002

- Fixed a bug: If you called DROP DATABASE for a database on which there simultaneously were running queries, the MySQL server could crash or hang. Crashes fixed, but a full fix has to wait some changes in the MySQL layer of code.
- Fixed a bug: on Windows one had to put the database name in lowercase for `DROP DATABASE` to work. Fixed in 3.23.49: case no longer matters on Windows. On Unix, the database name remains case sensitive.
- Fixed a bug: If one defined a non-`latin1` character set as the default character set, then definition of foreign key constraints could fail in an assertion failure in dict0crea.c, reporting an internal error 17.

C.4.37 Changes in MySQL/InnoDB-3.23.48, February 9, 2002

- Tuned the SQL optimizer to favor more often index searches over table scans.
- Fixed a performance problem when several large SELECT queries are run concurrently on a multiprocessor Linux computer. Large CPU-bound SELECT queries now also generally run faster on all platforms.
- If MySQL binary logging is used, `InnoDB` now prints after crash recovery the latest MySQL binary log file name and the position in that file (= byte offset) `InnoDB` was able to recover to. This is useful, for example, when resynchronizing a master and a slave database in replication.

- Added better error messages to help in installation problems.
- One can now recover also MySQL temporary tables which have become orphaned inside the `InnoDB` tablespace.
- `InnoDB` now prevents a FOREIGN KEY declaration where the signedness is not the same in the referencing and referenced integer columns.
- Fixed a bug: calling SHOW CREATE TABLE or SHOW TABLE STATUS could cause memory corruption and make `mysqld` to crash. Especially at risk was mysqldump, because it calls frequently SHOW CREATE TABLE.
- Fixed a bug: If on Unix you did an ALTER TABLE to an `InnoDB` table and simultaneously did queries to it, `mysqld` could crash with an assertion failure in row0row.c, line 474.
- Fixed a bug: If inserts to several tables containing an `AUTO_INCREMENT` column were wrapped inside one LOCK TABLES, `InnoDB` asserted in lock0lock.c.
- In 3.23.47 we permitted several NULLS in a UNIQUE secondary index. But CHECK TABLE was not relaxed: it reports the table as corrupt. CHECK TABLE no longer complains in this situation.
- Fixed a bug: on Sparc and other high-endian processors SHOW VARIABLES showed innodb_flush_log_at_trx_commit and other boolean-valued startup parameters always OFF even if they were switched on.
- Fixed a bug: If you ran mysqld-max-nt as a service on Windows NT/2000, the service shutdown did not always wait long enough for the `InnoDB` shutdown to finish.

C.4.38 Changes in MySQL/InnoDB-3.23.47, December 28, 2001

- Recovery happens now faster, especially in a lightly loaded system, because background checkpointing has been made more frequent.
- `InnoDB` now permits several similar key values in a `UNIQUE` secondary index if those values contain SQL `NULL` values. Thus the convention is now the same as in `MyISAM` tables.
- `InnoDB` gives a better row count estimate for a table which contains `BLOB` values.
- In a `FOREIGN KEY` constraint, `InnoDB` is now case-insensitive to column names, and in Windows also to table names.
- `InnoDB` permits a `FOREIGN KEY` column of `CHAR` type to refer to a column of `VARCHAR` type, and vice versa. MySQL silently changes the type of some columns between `CHAR` and `VARCHAR`, and these silent changes do not hinder `FOREIGN KEY` declaration any more.
- Recovery has been made more resilient to corruption of log files.
- Unnecessary statistics calculation has been removed from queries which generate a temporary table. Some ORDER BY and DISTINCT queries now run much faster.
- MySQL now knows that the table scan of an `InnoDB` table is done through the primary key. This saves a sort in some ORDER BY queries.
- The maximum key length of `InnoDB` tables is again restricted to 500 bytes. The MySQL interpreter is not able to handle longer keys.
- The default value of innodb_lock_wait_timeout was changed from infinite to 50 seconds, the default value of innodb_file_io_threads from 9 to 4.

C.4.39 Changes in MySQL/InnoDB-4.0.1, December 23, 2001

- `InnoDB` is the same as in 3.23.47.
- In 4.0.0 the MySQL interpreter did not know the syntax LOCK IN SHARE MODE. This has been fixed.
- In 4.0.0 multiple-table delete did not work for transactional tables. This has been fixed.

C.4.40 Changes in MySQL/InnoDB-3.23.46, November 30, 2001

- This is the same as 3.23.45.

C.4.41 Changes in MySQL/InnoDB-3.23.45, November 23, 2001

- This is a bugfix release.
- In versions 3.23.42-.44 when creating a table on Windows, you have to use lowercase letters in the database name to be able to access the table. Fixed in 3.23.45.
- `InnoDB` now flushes stdout and stderr every 10 seconds: If these are redirected to files, the file contents can be better viewed with an editor.
- Fixed an assertion failure in .44, in trx0trx.c, line 178 when you drop a table which has the .frm file but does not exist inside `InnoDB`.
- Fixed a bug in the insert buffer. The insert buffer tree could get into an inconsistent state, causing a crash, and also crashing the recovery. This bug could appear especially in large table imports or alterations.
- Fixed a bug in recovery: `InnoDB` could go into an infinite loop constantly printing a warning message that it cannot find free blocks from the buffer pool.
- Fixed a bug: when you created a temporary table of the `InnoDB` type, and then used ALTER TABLE to it, the MySQL server could crash.
- Prevented creation of MySQL system tables 'mysql.user', 'mysql.host', or 'mysql.db', in the `InnoDB` type.
- Fixed a bug which can cause an assertion failure in 3.23.44 in srv0srv.c, line 1728.

C.4.42 Changes in MySQL/InnoDB-3.23.44, November 2, 2001

- You can define foreign key constraints on `InnoDB` tables. An example: FOREIGN KEY (col1) REFERENCES table2(col2).
- You can create data files larger than 4GB in those file systems that allow it.
- Improved `InnoDB` monitors, including a new `innodb_table_monitor` which enables you to print the contents of the `InnoDB` internal data dictionary.
- `DROP DATABASE` now works also for `InnoDB` tables.
- Accent characters in the default character set latin1 are ordered according to the MySQL ordering.

  NOTE: If you are using latin1 and have inserted characters whose code is > 127 to an indexed CHAR column, you should run CHECK TABLE on your table when you upgrade to 3.23.43, and drop and reimport the table if CHECK TABLE reports an error!
- `InnoDB` calculates better table cardinality estimates.

- Change in deadlock resolution: in .43 a deadlock rolls back only the SQL statement, in .44 it rolls back the whole transaction.
- Deadlock, lock wait timeout, and foreign key constraint violations (no parent row, child rows exist) now return native MySQL error codes 1213, 1205, 1216, 1217, respectively.
- A new my.cnf parameter innodb_thread_concurrency helps in performance tuning in high concurrency environments.
- A new my.cnf option innodb_force_recovery helps you in dumping tables from a corrupted database.
- A new my.cnf option innodb_fast_shutdown speeds up shutdown. Normally `InnoDB` does a full purge and an insert buffer merge at shutdown.
- Raised maximum key length to 7000 bytes from a previous limit of 500 bytes.
- Fixed a bug in replication of `AUTO_INCREMENT` columns with multiline inserts.
- Fixed a bug when the case of letters changes in an update of an indexed secondary column.
- Fixed a hang when there are more than 24 data files.
- Fixed a crash when `MAX(col)` [884] is selected from an empty table, and *`col`* is a not the first column in a multi-column index.
- Fixed a bug in purge which could cause crashes.

C.4.43 Changes in MySQL/InnoDB-3.23.43, October 4, 2001

- This is essentially the same as InnoDB-3.23.42.

C.4.44 Changes in MySQL/InnoDB-3.23.42, September 9, 2001

- Fixed a bug which corrupted the table if the primary key of a > 8000-byte row was updated.
- There are now 3 types of `InnoDB` Monitors: innodb_monitor, innodb_lock_monitor, and innodb_tablespace_monitor. innodb_monitor now prints also buffer pool hit rate and the total number of rows inserted, updated, deleted, read.
- Fixed a bug in RENAME TABLE.
- Fixed a bug in replication with an auto-increment column.

C.4.45 Changes in MySQL/InnoDB-3.23.41, August 13, 2001

- Support for < 4GB rows. The previous limit was 8000 bytes.
- Use the doublewrite file flush method.
- Raw disk partitions supported as data files.
- `InnoDB` Monitor.
- Several hang bugs fixed and an `ORDER BY` bug (“Sort aborted”) fixed.

C.4.46 Changes in MySQL/InnoDB-3.23.40, July 16, 2001

- Only a few rare bugs fixed.

C.4.47 Changes in MySQL/InnoDB-3.23.39, June 13, 2001

- `CHECK TABLE` now works for `InnoDB` tables.
- A new `my.cnf` parameter `innodb_unix_file_flush_method` introduced. It can be used to tune disk write performance.
- An auto-increment column now gets new values past the transaction mechanism. This saves CPU time and eliminates transaction deadlocks in new value assignment.
- Several bugfixes, most notably the rollback bug in 3.23.38.

C.4.48 Changes in MySQL/InnoDB-3.23.38, May 12, 2001

- The new syntax `SELECT ... LOCK IN SHARE MODE` is introduced.
- `InnoDB` now calls `fsync()` after every disk write and calculates a checksum for every database page it writes or reads, which reveals disk defects.
- Several bugfixes.

C.5 MySQL Cluster Change History

Beginning with MySQL 4.1.14, MySQL Cluster changes for MySQL 4.1 Server releases can be found in the MySQL 4.1 Server Release Notes. For release notes for older releases of MySQL Cluster (before 4.1.14), see MySQL Cluster 4.1 Release Notes.

C.6 MySQL Connector/ODBC Change History

MySQL Connector/ODBC release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Connector/ODBC are located at MySQL Connector/ODBC Release Notes.

C.7 MySQL Connector/Net Change History

MySQL Connector/Net release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Connector/Net are located at MySQL Connector/Net Release Notes.

C.8 MySQL Connector/J Change History

MySQL Connector/J release notes are no longer published in the MySQL Reference Manual.

Release notes for the changes in each release of MySQL Connector/J are located at MySQL Connector/J Release Notes.

Appendix D Restrictions and Limits

Table of Contents

The discussion here describes restrictions that apply to the use of MySQL features such as subqueries.

D.1 Restrictions on Subqueries

- Known bug: If you compare a `NULL` value to a subquery using `ALL`, `ANY`, or `SOME`, and the subquery returns an empty result, the comparison might evaluate to the nonstandard result of `NULL` rather than to `TRUE` or `FALSE`. This is fixed in MySQL 5.0.36 and 5.1.16.
- A subquery's outer statement can be any one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`.
- Subquery optimization for `IN` is not as effective as for the `=` operator or for the `IN(value_list)` [784] operator.

  A typical case for poor `IN` subquery performance is when the subquery returns a small number of rows but the outer query returns a large number of rows to be compared to the subquery result.

  The problem is that, for a statement that uses an `IN` subquery, the optimizer rewrites it as a correlated subquery. Consider the following statement that uses an uncorrelated subquery:

  ```
  SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
  ```

  The optimizer rewrites the statement to a correlated subquery:

  ```
  SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t1.a);
  ```

  If the inner and outer queries return *M* and *N* rows, respectively, the execution time becomes on the order of O(*M*×*N*), rather than O(*M*+*N*) as it would be for an uncorrelated subquery.

  An implication is that an `IN` subquery can be much slower than a query written using an `IN(value_list)` [784] operator that lists the same values that the subquery would return.
- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

  ```
  DELETE FROM t WHERE ... (SELECT ... FROM t ...);
  UPDATE t ... WHERE col = (SELECT ... FROM t ...);
  {INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
  ```

  Exception: The preceding prohibition does not apply if you are using a subquery for the modified table in the `FROM` clause. Example:

  ```
  UPDATE t ... WHERE col = (SELECT * FROM (SELECT ... FROM t...) AS _t ...);
  ```

Here the result from the subquery in the FROM clause is stored as a temporary table, so the relevant rows in t have already been selected by the time the update to t takes place.

- Row comparison operations are only partially supported:
  - For *expr* IN (*subquery*), *expr* can be an *n*-tuple (specified using row constructor syntax) and the subquery can return rows of *n*-tuples.
  - For *expr op* {ALL|ANY|SOME} (*subquery*), *expr* must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

  In other words, for a subquery that returns rows of *n*-tuples, this is supported:

```
(val_1, ..., val_n) IN (subquery)
```

  But this is not supported:

```
(val_1, ..., val_n) op {ALL|ANY|SOME} (subquery)
```

  The reason for supporting row comparisons for IN but not for the others is that IN is implemented by rewriting it as a sequence of = [782] comparisons and AND [787] operations. This approach cannot be used for ALL, ANY, or SOME.

- Row constructors are not well optimized. The following two expressions are equivalent, but only the second can be optimized:

```
(col1, col2, ...) = (val1, val2, ...)
col1 = val1 AND col2 = val2 AND ...
```

- Subqueries in the FROM clause cannot be correlated subqueries. They are materialized (executed to produce a result set) before evaluating the outer query, so they cannot be evaluated per row of the outer query.

- MySQL does not support LIMIT in subqueries for certain subquery operators:

```
mysql> SELECT * FROM t1
    ->   WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1);
ERROR 1235 (42000): This version of MySQL doesn't yet support
 'LIMIT & IN/ALL/ANY/SOME subquery'
```

- The optimizer is more mature for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as a join.

  An exception occurs for the case where an IN subquery can be rewritten as a SELECT DISTINCT join. Example:

```
SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condition);
```

  That statement can be rewritten as follows:

```
SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condition;
```

  But in this case, the join requires an extra DISTINCT operation and is not more efficient than the subquery.

- MySQL permits a subquery to refer to a stored function that has data-modifying side effects such as inserting rows into a table. For example, if `f()` inserts rows, the following query can modify data:

  ```
  SELECT ... WHERE x IN (SELECT f() ...);
  ```

  This behavior is nonstandard (not permitted by the SQL standard). In MySQL, it can produce indeterminate results because `f()` might be executed a different number of times for different executions of a given query depending on how the optimizer chooses to handle it.

  For replication, one implication of this indeterminism is that such a query can produce different results on the master and its slaves.

- Possible future optimization: MySQL does not rewrite the join order for subquery evaluation. In some cases, a subquery could be executed more efficiently if MySQL rewrote it as a join. This would give the optimizer a chance to choose between more execution plans. For example, it could decide whether to read one table or the other first.

  Example:

  ```
  SELECT a FROM outer_table AS ot
  WHERE a IN (SELECT a FROM inner_table AS it WHERE ot.b = it.b);
  ```

  For that query, MySQL always scans `outer_table` first and then executes the subquery on `inner_table` for each row. If `outer_table` has a lot of rows and `inner_table` has few rows, the query probably will not be as fast as it could be.

  The preceding query could be rewritten like this:

  ```
  SELECT a FROM outer_table AS ot, inner_table AS it
  WHERE ot.a = it.a AND ot.b = it.b;
  ```

  In this case, we can scan the small table (`inner_table`) and look up rows in `outer_table`, which will be fast if there is an index on `(ot.a,ot.b)`.

- Possible future optimization: A correlated subquery is evaluated for each row of the outer query. A better approach is that if the outer row values do not change from the previous row, do not evaluate the subquery again. Instead, use its previous result.

- Possible future optimization: A subquery in the `FROM` clause is evaluated by materializing the result into a temporary table, and this table does not use indexes. This does not allow the use of indexes in comparison with other tables in the query, although that might be useful.

- Possible future optimization: If a subquery in the `FROM` clause resembles a view to which the merge algorithm can be applied, rewrite the query and apply the merge algorithm so that indexes can be used. The following statement contains such a subquery:

  ```
  SELECT * FROM (SELECT * FROM t1 WHERE t1.t1_col) AS _t1, t2 WHERE t2.t2_col;
  ```

  The statement can be rewritten as a join like this:

  ```
  SELECT * FROM t1, t2 WHERE t1.t1_col AND t2.t2_col;
  ```

  This type of rewriting would provide two benefits:

  - It avoids the use of a temporary table for which no indexes can be used. In the rewritten query, the optimizer can use indexes on `t1`.
  - It gives the optimizer more freedom to choose between different execution plans. For example, rewriting the query as a join enables the optimizer to use `t1` or `t2` first.

- Possible future optimization: For `IN`, `= ANY`, `<> ANY`, `= ALL`, and `<> ALL` with uncorrelated subqueries, use an in-memory hash for a result or a temporary table with an index for larger results. Example:

```
SELECT a FROM big_table AS bt
WHERE non_key_field IN (SELECT non_key_field FROM table WHERE condition)
```

  In this case, we could create a temporary table:

```
CREATE TABLE t (key (non_key_field))
(SELECT non_key_field FROM table WHERE condition)
```

  Then, for each row in `big_table`, do a key lookup in `t` based on `bt.non_key_field`.

D.2 Restrictions on Character Sets

- Identifiers are stored in `mysql` database tables (`user`, `db`, and so forth) using `utf8`, but identifiers can contain only characters in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted in identifiers.
- The `ucs2` character sets has the following restrictions:
  - It cannot be used as a client character set, which means that it does not work for `SET NAMES` or `SET CHARACTER SET`. (See Section 9.1.4, “Connection Character Sets and Collations”.)
  - It is currently not possible to use `LOAD DATA INFILE` to load data files that use this character set.
  - `FULLTEXT` indexes cannot be created on a column that this character set. However, you can perform `IN BOOLEAN MODE` searches on the column without an index.
- The `REGEXP` [808] and `RLIKE` [808] operators work in byte-wise fashion, so they are not multi-byte safe and may produce unexpected results with multi-byte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

D.3 Limits in MySQL

This section lists current limits in MySQL 4.1.

D.3.1 Limits of Joins

In MySQL 4.1, the maximum number of tables that can be referenced in a single join is 61. This also applies to the number of tables that can be referenced in the definition of a view.

D.3.2 The Maximum Number of Columns Per Table

There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table. The exact limit depends on several interacting factors, listed in the following discussion.

- Every table has a maximum row size of 65,535 bytes. This maximum applies to all storage engines, but a given engine might have additional constraints that result in a lower effective maximum row size.

  The maximum row size constrains the number of columns because the total width of all columns cannot exceed this size. For example, `utf8` characters require up to three bytes per character, so for a `CHAR(255) CHARACTER SET utf8` column, the server must allocate 255 × 3 = 765 bytes per value. Consequently, a table cannot contain more than 65,535 / 765 = 85 such columns.

  Storage for variable-length columns includes length bytes, which are assessed against the row size. For example, a `VARCHAR(255) CHARACTER SET utf8` column takes two bytes to store the length of the value, so each value can take up to 767 bytes.

  `BLOB` and `TEXT` columns count from one to four plus eight bytes each toward the row-size limit because their contents are stored separately.

  Declaring columns `NULL` can reduce the maximum number of columns permitted. `NULL` columns require additional space in the row to record whether their values are `NULL`.

  For `MyISAM` and `ISAM` tables, each `NULL` column takes one bit extra, rounded up to the nearest byte. The maximum row length in bytes can be calculated as follows:

  ```
  row length = 1
               + (sum of column lengths)
               + (number of NULL columns + delete_flag + 7)/8
               + (number of variable-length columns)
  ```

  `delete_flag` is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *`delete_flag`* is 0 for dynamic tables because the flag is stored in the dynamic row header.

  These calculations do not apply for `InnoDB` tables, for which storage size is no different for `NULL` columns than for `NOT NULL` columns.

- Each table has an `.frm` file that contains the table definition. The server uses the following expression to check some of the table information stored in the file against an upper limit of 64KB:

  ```
  if (info_length+(ulong) create_fields.elements*FCOMP+288+
      n_length+int_length+com_length > 65535L || int_count > 255)
  ```

  The portion of the information stored in the `.frm` file that is checked against the expression cannot grow beyond the 64KB limit, so if the table definition reaches this size, no more columns can be added.

  The relevant factors in the expression are:

  - `info_length` is space needed for “screens.” This is related to MySQL's Unireg heritage.
  - `create_fields.elements` is the number of columns.
  - `FCOMP` is 17.
  - `n_length` is the total length of all column names, including one byte per name as a separator.
  - `int_length` is related to the list of values for `ENUM` and `SET` columns.
  - `com_length` is the total length of column comments.

Thus, using long column names can reduce the maximum number of columns, as can the inclusion of `ENUM` or `SET` columns, or use of column comments.

- Individual storage engines might impose additional restrictions that limit table column count. Examples:
  - `InnoDB` permits no more than 1000 columns.
  - `InnoDB` restricts row size to something less than half a database page (approximately 8000 bytes), not including `VARBINARY`, `VARCHAR`, `BLOB`, or `TEXT` columns.

D.3.3 Windows Platform Limitations

The following limitations apply to use of MySQL on the Windows platform:

- **Number of file descriptors**

  The number of open file descriptors on Windows is limited to a maximum of 2048, which may limit the ability to open a large number of tables simultaneously. This limit is due not to Windows but to C runtime library compatibility functions used to open files on Windows that use the POSIX compatibility layer.

  This limitation will also cause problems if you try to set `open_files_limit` to a value greater than the 2048 file limit.

- **Process memory**

  On Windows 32-bit platforms it is not possible by default to use more than 2GB of RAM within a single process, including MySQL. This is because the physical address limit on Windows 32-bit is 4GB and the default setting within Windows is to split the RAM between kernel (2GB) and user/applications (2GB).

  You can increase this limit to 3GB by specifying the `/3GB` option in the `boot.ini` file. This changes the kernel/application memory split to 1GB and 3GB respectively. This boot option is available on Windows XP, Windows Server 2003, and Windows Server 2008.

  Some versions of Windows have a boot time setting to enable larger applications by reducing the kernel application. Alternatively, to use more than 2GB, use a 64-bit version of Windows.

- **File system aliases**

  When using `MyISAM` tables, you cannot use aliases within Windows link to the data files on another volume and then link back to the main MySQL `datadir` [385] location.

  This facility is often used to move the data and index files to a RAID or other fast solution, while retaining the main `.frm` files in the default data directory configured with the `datadir` [385] option.

- **Limited number of ports**

  Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Note that ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

  For more information about this problem, see http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

- **Concurrent reads**

MySQL depends on the `pread()` and `pwrite()` calls to be able to mix `INSERT` and `SELECT`. Currently, we use mutexes to emulate `pread()`/`pwrite()`. We will, in the long run, replace the file level interface with a virtual interface so that we can use the `readfile()`/`writefile()` interface on NT, 2000, and XP to get more speed. The current implementation limits the number of open files that MySQL can use to 2,048 (1,024 before MySQL 4.0.19), which means that you cannot run as many concurrent threads on NT, 2000, XP, and 2003 as on Unix.

This problem is fixed in MySQL 5.5.

- **Blocking read**

  MySQL uses a blocking read for each connection. That has the following implications if named-pipe connections are enabled:

  - A connection is not disconnected automatically after eight hours, as happens with the Unix version of MySQL.
  - If a connection hangs, it is impossible to break it without killing MySQL.
  - `mysqladmin kill` does not work on a sleeping connection.
  - `mysqladmin shutdown` cannot abort as long as there are sleeping connections.

  These problems are fixed in MySQL 5.1. (Bug #31621)

- **`ALTER TABLE`**

  While you are executing an `ALTER TABLE` statement, the table is locked from being used by other threads. This has to do with the fact that on Windows, you cannot delete a file that is in use by another thread. In the future, we may find some way to work around this problem.

- **`DROP TABLE`**

  `DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` handler does the table mapping hidden from the upper layer of MySQL. Because Windows does not permit you to drop files that are open, you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table. We will fix this at the same time we introduce views.

- **`DATA DIRECTORY` and `INDEX DIRECTORY`**

  The `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows, because MySQL does not support Windows symbolic links. These options also are ignored on systems that have a nonfunctional `realpath()` call.

- **`DROP DATABASE`**

  You cannot drop a database that is in use by another session.

- **Case-insensitive names**

  File names are not case sensitive on Windows, so MySQL database and table names are also not case sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See Section 8.2.2, “Identifier Case Sensitivity”.

- **Directory and file names**

On Windows, MySQL Server supports only directory and file names that are compatible with the current ANSI code pages. For example, the following Japanese directory name will not work in the Western locale (code page 1252):

```
datadir="C:/私たちのプロジェクトのデータ"
```

The same limitation applies to directory and file names referred to in SQL statements, such as the data file path name in `LOAD DATA INFILE`.

- **The "`\`" path name separator character**

  Path name components in Windows are separated by the "`\`" character, which is also the escape character in MySQL. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, use Unix-style file names with "`/`" characters:

  ```
  mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
  ```

  Alternatively, you must double the "`\`" character:

  ```
  mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
  ```

- **Problems with pipes**

  Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z` / `CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

  This is a problem mainly when you try to apply a binary log as follows:

  ```
  shell> mysqlbinlog binary_log_file | mysql --user=root
  ```

  If you have a problem applying the log and suspect that it is because of a `^Z` / `CHAR(24)` character, you can use the following workaround:

  ```
  shell> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql
  shell> mysql --user=root --execute "source /tmp/bin.sql"
  ```

  The latter command also can be used to reliably read in any SQL file that may contain binary data.

- **Killing MySQL from the Task Manager**

  On Windows 95, you cannot kill MySQL from the Task Manager or with the shutdown utility. You must stop it with `mysqladmin shutdown` or the `NET STOP ...` command.

- **Windows 95 and threads**

  Windows 95 leaks about 200 bytes of main memory for each thread creation. Each connection in MySQL creates a new thread, so you should not run `mysqld` for an extended time on Windows 95 if your server handles many connections! Newer versions of Windows do not suffer from this bug.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z