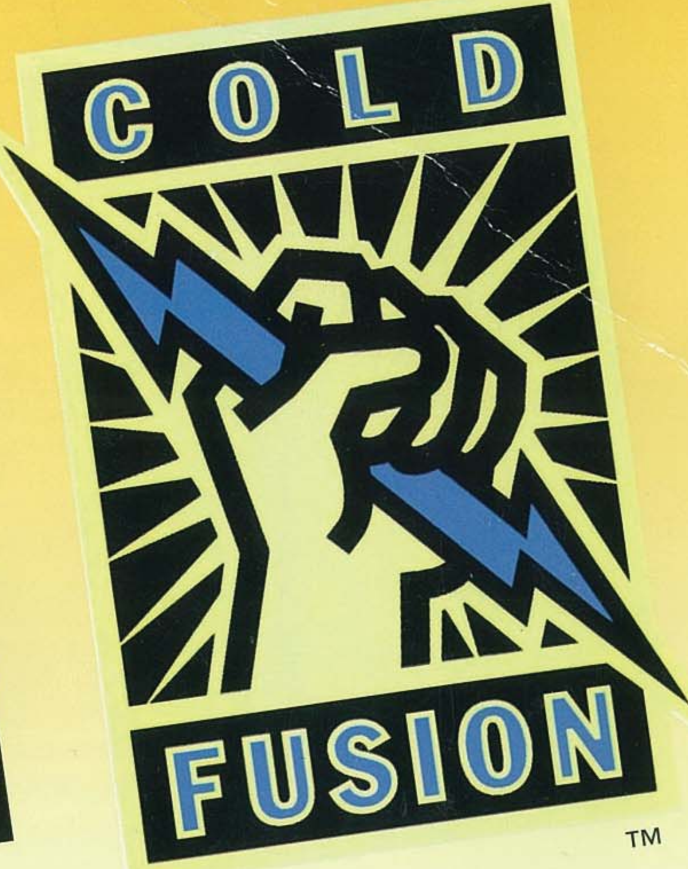
Everything you need to create powerful Web-based applications

"The most complete and compelling set of materials and information on Cold Fusion."

JJ Allaire
founder of Allaire Corp.
and creator of Cold Fusion



the Cold Fusion Web Database Construction Kit

CD-ROM includes Cold Fusion Single-User Version

Ben Forta
with
Steven Drucker
and David Watts

ALLAIRE™

que®

001

ServiceNow, Inc.'s
Exhibit No. 1005

About two years ago, when the Web was still in its infancy, I had the privilege of being inspired by a small but strongly emergent group of developers. These developers came from all walks of life; some were designers or college students, many were computer professionals of some sort or another. It didn't matter where they came from, or what experience they had. What was common about them was that they were all fanatics. They had come into new careers and new visions because of the simplicity and beauty of the Web, and they were certain that the Web was going to change the world.

This group of fanatics inspired the creation of Cold Fusion, which aimed to be the first tool available to enable any Web developer to build interactive Web applications—to actually use the Web to build software—without having to have been a professional programmer. By now it is commonly understood that the Web is transforming computing and society, and that more than ever tools are needed for the average person to be able to build these systems.

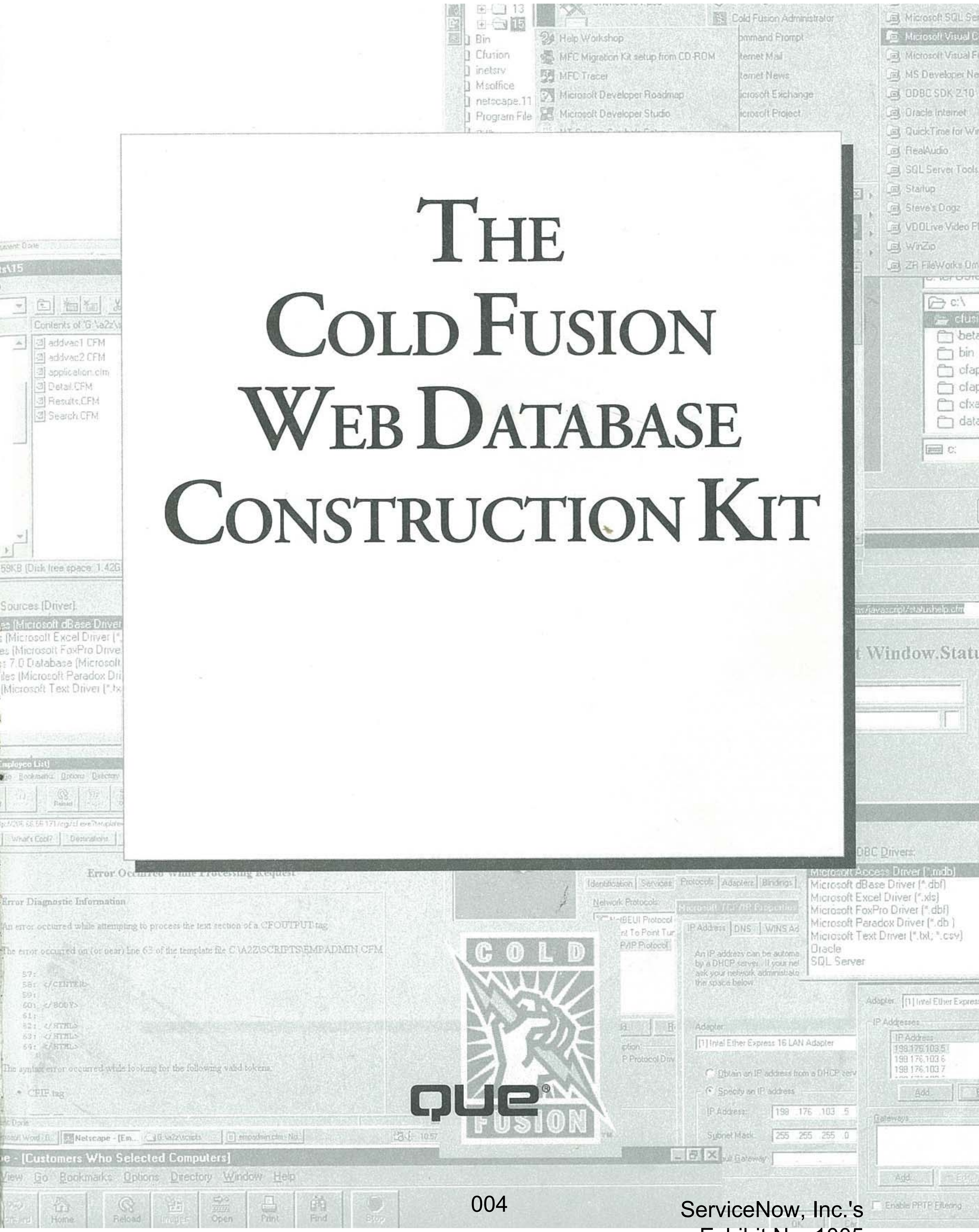
We're now shipping Cold Fusion 2.0, our third major release of this leading Internet product. As I hope you'll find from this book, Cold Fusion 2.0 brings together unparalleled features and functionality in an incredibly useable environment for Web developers. Using our server-side markup language, CFML, you should be able to rapidly begin building Web applications which you never dreamed were possible, or which you thought could only be built by advanced professional programmers.

With this book, Ben Forta has brought together the most complete and compelling set of materials and information on Cold Fusion, and has presented it in a logical and readable fashion.

As a learning tool for beginners, the book contains the key information you'll need to get started, including the basics of Cold Fusion, an explanation of databases and application design, and in-depth information on using SQL. For experienced Cold Fusion developers, the depth of information about CFML, including tips and tricks that I didn't even know of myself, will make it a phenomenal reference. And for those of you who want to be on the cutting-edge of Web applications, advanced materials on the CFAPI, Java integration, and dynamic VRML allow you to push the limits of the Web platform.

With this book, we welcome you to a community of thousands of developers worldwide who are powering their Web applications with Cold Fusion.

J.J. Allaire
Founder of Allaire Corp., and creator of Cold Fusion

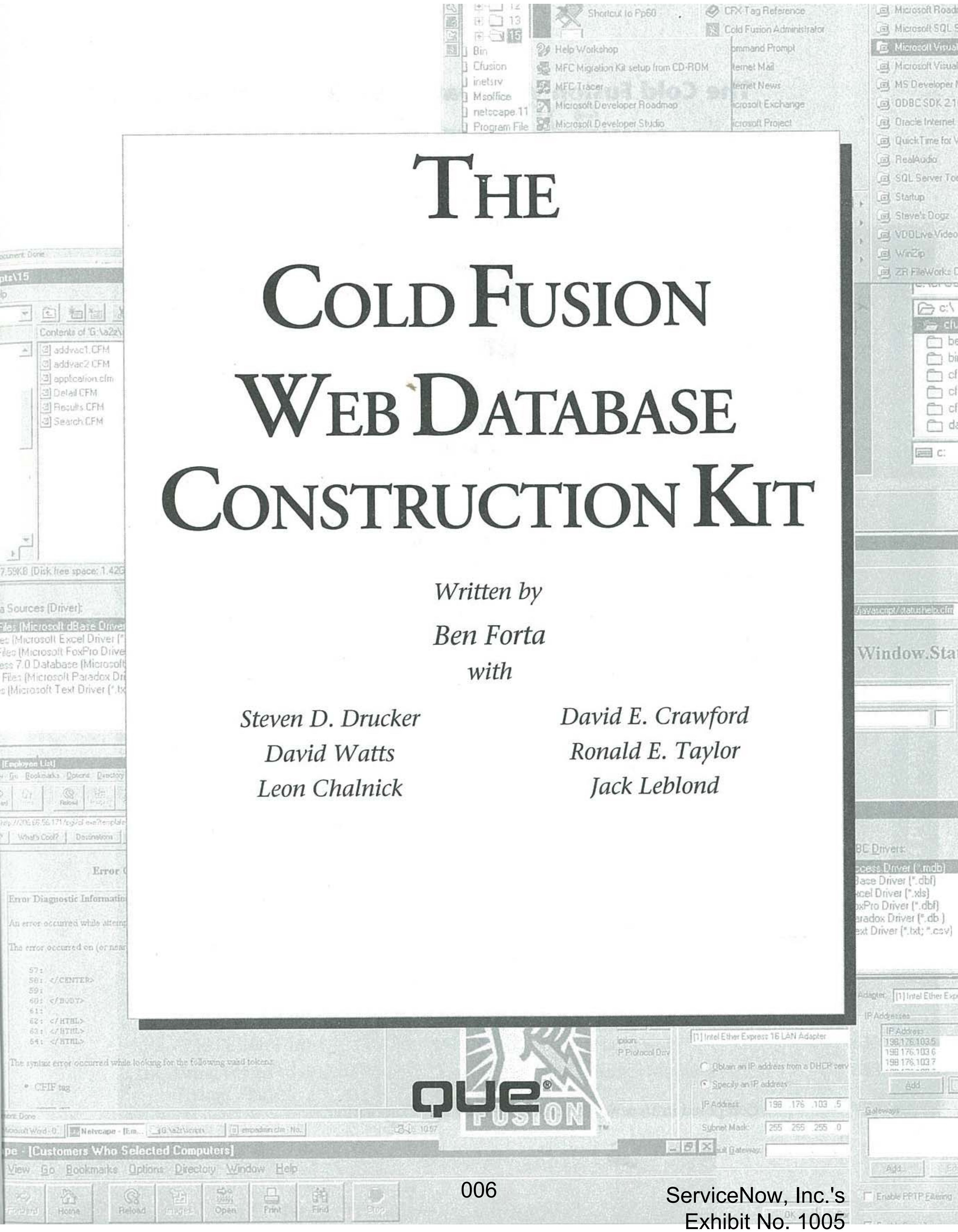


The Cold Fusion Web Database Construction Kit

que®

004

ServiceNow, Inc.'s
Exhibit No. 1005



THE COLD FUSION WEB DATABASE CONSTRUCTION KIT

Written by

Ben Forta

with

Steven D. Drucker
David Watts
Leon Chalnick
David E. Crawford
Ronald E. Taylor
Jack Leblond

que®

The Cold Fusion Web Database Construction Kit

Library of Congress Catalog No.: 96-071452

ISBN: 0-7897-0970-8

99 98 97 6 5 4 3 2 1

Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit number, the number of the book's printing. For example, a printing code of 97-1 shows that the first printing of the book occurred in 1997.

Screen reproductions in this book were created using Collage Plus from Inner Media, Inc., Hollis, NH.

Composed in *Stone Serif* and *MCPdigital* by Que Corporation

Contents at a Glance

Contents

Chapter 1

Why Cold Fusion?

Ben Forta

In this chapter:

- **What is Cold Fusion?**
- **Enhancing your Internet and Intranet**
- **Dynamic publishing rather than static text**
- **Complete Development Environment**

Over the past few years, the World Wide Web has become one of the most accepted mediums for publishing information to the masses. Web pages that combine text, images, animation, and even multimedia, have become commonplace.

Cold Fusion is a complete development environment that provides the tools you need to take your Web pages to a whole new dimension. With Cold Fusion, you can create dynamic data driven Web sites instead of creating static pages of fixed text and images. You can collect data, read and write information in other applications, and even create full blown applications of your own.

In this chapter, you learn why your Web site should be powered by Cold Fusion.

Introducing Cold Fusion

The fact that you are reading this book suggests that you are interested in publishing information on the World Wide Web and that you are part of a growing number of developers interested in expanding the capabilities of their Web sites beyond simple publishing.

What kind of capabilities?

There are now over 1/4 million Web sites that attract millions of visitors daily. Most Web sites are being used as electronic replacements for newspapers, magazines, brochures, and bulletin boards. The Web offers ways to enhance these publications using audio, images, animation, multimedia, and even virtual reality.

No one will dispute that these sites add value to the Net because information is knowledge, and knowledge is power. All this information is available at your fingertips (literally). Web sites, however, are capable of being much more than electronic versions of paper publications because of the underlying technology that makes the Web tick. Users can interact with you and your company, collect and process mission critical information in real time, provide new levels of user support, and much more.

The Web is not merely the electronic equivalent of a newspaper or magazine—it is a communication medium that is only limited by the lack of innovation and creativity of Web site designers.

To help illustrate what Cold Fusion is and where it fits into your Web site strategy, let's look at a few of the more impressive and innovative sites on the World Wide Web:

- **Dell Computer Corporation (http://www.dell.com)**

  Dell is a leading vendor of mail order computers. Their Web site, like many others, enables you to shop for a new computer online. The big difference is that you may customize the computer online. You are presented with a typical configuration with a price tag attached. You may click on any of the components or peripherals to add or change them. Click on the hard drive line item, for example, and you'll be presented with other hard drive options and how they will effect the price. As soon as any changes are made to the configuration, the price tag is updated automatically.

- **Federal Express (http://www.fedex.com)**

  The FedEx site hosts a whole array of impressive and innovative features. The most impressive is the online package tracking system. To test the system, I deposited a package (containing chapters that are now part of this book) in a drop box at 9:55am. At 10:05am I checked the status of the package on the FedEx Web site. In less time than it took to click the search button, I was informed that my package was picked up at 10:02am from the Southfield, MI drop box. FedEx did not design a complete package tracking system for the Web but cleverly linked their Web site to an already existing application. In doing so, they provide superior customer support and lowered their real time phone support costs.

- **Ticketmaster (http://www.ticketmaster.com)**

  The Ticketmaster Web site is built around a massive database of every entertainment event in every city at every venue in the United States. The database can be searched by date, artist, event type, state, city, venue, category, and more. The site even contains seating maps of venues hosting events.

These selected sites are truly taking advantage of the World Wide Web.

What is Cold Fusion?

Until recently, developing sites (like the ones mentioned earlier) was a difficult process. Writing custom Web-based applications was a job for experienced programmers only. A good working knowledge of UNIX was a prerequisite, and experience with traditional development or scripting languages was a must.

But all that has changed. Allaire's Cold Fusion enables you to create sites every bit as powerful and as capable as the ones listed earlier, without a long and painful learning curve.

So, what is Cold Fusion? Simply put, Cold Fusion is a Web application development tool that enables the rapid creation of interactive, dynamic, and information-rich Web sites.

Cold Fusion does not require coding in traditional programming languages. Instead, you create applications by extending your standard HTML files with high level formatting functions, conditional operators, and database commands. These commands are instructions to the Cold Fusion processor and form the building blocks on which to build industrial strength applications.

This method of creating Web applications has significant advantages over conventional application development.

- Cold Fusion applications can be developed rapidly because no coding is required other than use of simple HTML style tags.
- Cold Fusion applications are easy to test and roll out.
- The Cold Fusion language contains all the processing and formatting functions you'll need (and the ability to create your own functions if you really run into a dead end).
- Cold Fusion applications are easy to maintain because there is no compilation or linking step, so the files you create are the files used by Cold Fusion.
- Cold Fusion provides all the tools you need to be able to troubleshoot and debug applications.
- Cold Fusion comes with all the hooks needed to link to almost any database application.
- Cold Fusion is fast, thanks to its service-based architecture.

Cold Fusion and Your Intranet

Although all the examples mentioned so far have been Internet sites, the benefits of Cold Fusion apply to Intranets, too.

Most companies have masses of information stored in different systems. Users often don't know what information is available or how to access it even if they do.

Cold Fusion bridges the gap between existing and legacy applications and your employees and empowers them with the tools to work more efficiently.

▶▶ Chapter 2, "Introduction to Cold Fusion," discusses Intranets in detail.

The Dynamic Page Advantage

Linking your Web site to live data is a tremendous advantage, but the benefits of database interaction go beyond extending the capabilities of your Web site.

With Cold Fusion you can create dynamic, data-driven Web pages. Dynamic Web pages are becoming the norm, and for a good reason. Consider the following:

- **Static Web pages**

  Static Web pages are made up of text and images, and HTML formatting tags. These pages are manually created and maintained so when information changes, so must the page. This usually involves loading the page into an editor, making the changes, reformatting text if needed, and then saving the file. Of course, not everyone in the organization can make these changes. The Webmaster, or Web design team, is responsible for maintaining the site and implementing all changes and enhancements. This often means that by the time information finally makes it onto the Web site, it is out of date.

- **Dynamic Web pages**

  Dynamic Web pages contain very little actual text. Instead, they pull needed information from other applications. Dynamic Web pages communicate with databases to extract employee directory information, spreadsheets to display accounting figures, client-server database management systems to interact with order processing applications, and more. Because a database *already* exists, why re-create it for Web page publication?

Cold Fusion provides you with a full range of database interaction functions to create complete dynamic, data-driven Web pages. The features include:

- The ability to query existing database applications for data.
- The ability to create dynamic queries facilitating more flexible data retrieval.
- The ability to execute stored procedures in databases that support them.
- The ability to execute conditional code on-the-fly, to customize responses for specific situations.
- The ability to enhance the standard HTML form capabilities with data validation functions.
- The ability to dynamically populate form elements.
- The ability to customize the display of dates, times, and currency values with formatting functions.
- The ability to ease the creation of data entry and data drill-down applications with "Wizards."

Powered by Cold Fusion

You were probably planning to use Cold Fusion to solve a particular problem or to fill a specific need. While this book helps you solve that problem, I hope that your mind is now racing and beginning to envision just what else Cold Fusion can do for your Web site.

Cold Fusion is a remarkable tool that is easy to learn, fun to use, and powerful enough to create real-world Web based applications.

With a minimal investment of your time, your Web site can be powered by Cold Fusion.

From Here...

In this chapter you were introduced to Cold Fusion. Cold Fusion is a remarkable tool that helps you create powerful Web based applications.

- Chapter 2, "Introduction to Cold Fusion," teaches you how Cold Fusion works and discusses the underlying technologies that Cold Fusion uses.
- Chapter 3, "Prerequisites," teaches the basic requirements that need to be in place to successfully install and run Cold Fusion.
- Chapter 4, "Installing and Administering Cold Fusion," walks you through the complete Cold Fusion installation process.

Chapter 8

Introduction to SQL

Ben Forta

In this chapter:

In Chapter 7, "Creating Databases and Tables," you created a database and tables for A2Z Books. In this chapter, you will learn how to retrieve data from those tables.

Structured Query Language is the language used by Cold Fusion for all database interaction. To harness the power of Cold Fusion you need a thorough understanding of SQL. The SQL statement you use to retrieve data from a table is the SELECT statement. Using SELECT, you also can sort and filter data to retrieve exactly the information you need.

Introducing SQL, the Structured Query Language

SQL, pronounced *sequel*, is an acronym for Structured Query Language. SQL is a language you use to access and manipulate data in a relational database. It is designed to be both easy to learn and extremely powerful, and its mass acceptance by so many database vendors proves that it has succeeded in both.

In 1970, Dr. E. F. Codd, the man credited with being the father of the relational database, described a universal language for data access. In 1974, engineers at IBM's San Jose Research Center created the Structured English Query Language, or SEQUEL, built on Codd's ideas. This language was incorporated into System R, IBM's pioneering relational database system.

Toward the end of the 1980s, two of the most important standards bodies, the American National Standards Institute (ANSI) and the International Standards Organization (ISO), published SQL standards, opening the door to mass acceptance. With these standards in place, SQL was poised to become the *de facto* standard used by every major database vendor.

Although SQL has evolved a great deal since its early SEQUEL days, the basic language concepts and its founding premises have remained the same. The beauty of SQL is its simplicity. But don't let that simplicity deceive you. SQL is a powerful language, and it encourages you to be creative in your problem solving. You can almost always find more than one way to perform a complex query or to extract desired data. Each solution has pros and cons, and no solution is explicitly right or wrong.

Before you panic at the thought of learning a new language, let me reassure you that SQL really is easy to learn. In fact, you need to learn only four statements to be able to perform almost all the data manipulation that you will need on a regular basis. Table 8.1 lists these statements.

Table 8.1 SQL-Based Data Manipulation Statements

Statement	Description
SELECT	Query a table for specific data.
INSERT	Add new data to a table.
UPDATE	Update existing data in a table.
DELETE	Remove data from a table.

Each of these statements takes one or more keywords as parameters. By combining different statements and keywords, you can manipulate your data in as many different ways as you can imagine.

Cold Fusion provides you with all the tools you need to create Web-based interaction to your databases. Cold Fusion itself, though, has no built-in database. Instead, it communicates with whatever database you select, passing updates and requests and returning query results.

Introducing ODBC, Open Database Connectivity

The communication between Cold Fusion and the database is via a database interface called Open Database Connectivity, or ODBC. ODBC is a standard application programming interface (API) for accessing information from different database systems and different storage formats. The purpose of ODBC is to enable you to access a diverse selection of databases and data formats without having to learn the features and peculiarities of each. ODBC provides a layer of abstraction, accomplished using database drivers, between your client application and the underlying database. The database drivers create a database-independent environment, as illustrated in Figure 8.1. This way, you can write one program and have it work with almost any major database system.

Of course, differences exist between database systems. Microsoft SQL Server, for example, requires you to log in to the database server before you are able to manipulate any data. Based on your login, you are granted or denied access to specific tables or other objects. Microsoft Access, on the other hand, has no concept of login-based security. If you have access to the data file (the MDB file), then you have full access to all data in it.

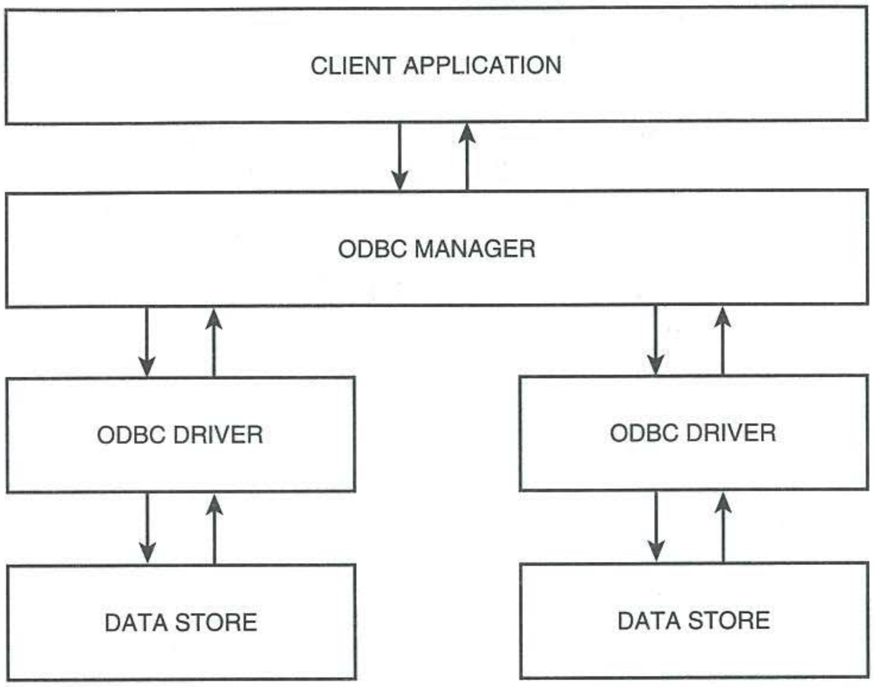
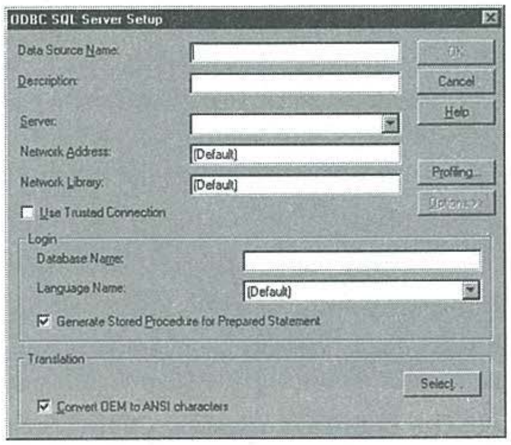


Figure 8.1 ODBC creates a database-independent development environment.

There are other differences, too. To access Microsoft SQL Server, your client application must know the address of the server. This might be an IP address or an NT Server name. To use Microsoft Access data files, you just need to know the drive and path to the data file.

Part of the job of ODBC is to hide these differences from your client application. And to accomplish this, each ODBC driver has its own configuration options. When you select the SQL Server ODBC driver, you are asked for a server name, a server login name, and a password, as shown in Figure 8.2. When you select the Access ODBC driver, you are prompted for a file path, as shown in Figure 8.3.



ODBC SQL Server Setup

Data Source Name:	
Description:	
Server:	
Network Address:	(Default)
Network Library:	(Default)

☐ Use Trusted Connection

Login

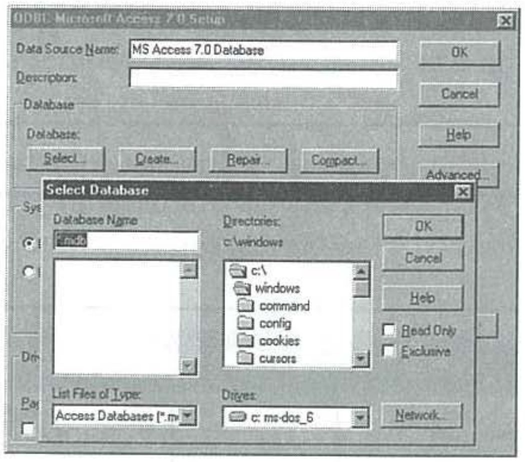
Database Name:	
Language Name:	(Default)

☑ Generate Stored Procedure for Prepared Statement

Translation

☑ Convert OEM to ANSI characters

Figure 8.2 The Microsoft SQL Server ODBC driver prompts you for login information.


ODBC Microsoft Access 7.0 Setup

Data Source Name: MS Access 7.0 Database

Description:

Select Database

Database Name: *.mdb

Directories: c:\windows

List Files of Type: Access Databases (*.m

Drives: c: ms-dos_6

Figure 8.3 The Microsoft Access ODBC driver prompts you for the file path to the Access data file.

This way, your client software can load any ODBC driver and connect to a database. The ODBC driver you select will handle opening the database, whether it's opening a network file or logging in to a server. All your client software knows is that it must connect to a database; the details of how this process occurs are all hidden.

The ODBC Story

ODBC was created in an effort to allow Microsoft Excel, Microsoft's popular spreadsheet program, to access diverse data stores.

In April 1988, Microsoft's Kyle Geiger proposed a model that used database drivers to isolate the native data types of different database applications. This model, in conjunction with a standard application interface, would allow client software to communicate with any message store. To access a particular data store, all that would be required is a driver designed specifically for that data store.

While Geiger worked on his proposal, engineers at DEC, Lotus, and Sybase were working on much the same ideas. The four companies joined forces, and between 1988 and 1992 they helped shape the specification.

The original name for this project was Microsoft Data Access API. In early 1989, the effort was renamed Open SQL, and then in the summer of 1989, it was renamed again to SQL Connectivity. Finally, in the winter of 1992, the name was changed one last time to Open Database Connectivity, or ODBC.

The beta version of ODBC 1.0 was released in March 1992, and in September 1992, version 1.0 finally was released. Shortly thereafter, in October 1992, the specification was reviewed and accepted by the ANSI SQL committee.

ODBC itself is not a language; the language used by ODBC is SQL. Part of the magic of the ODBC database driver is that it understands SQL and converts it to whatever is appropriate for that specific database. This way, you can use SQL commands to work with xBASE-based databases, such as Microsoft FoxPro and Borland dBASE, even though they have an entirely different native language.

Herein lies the power of ODBC. The combination of database independence and a common standard language grants ODBC clients. . . a tremendous level of freedom—freedom to use any database they want, freedom to use different databases for different tasks seamlessly and simultaneously, and the freedom to concentrate on application development without having to learn database-specific languages and API's.

ODBC and Cold Fusion

Cold Fusion is an ODBC client. ODBC enables you to use Cold Fusion with whatever database you choose. If you're using Microsoft Access, then Cold Fusion uses the Access driver; if you're using Oracle, then the Oracle ODBC driver is used instead. You can even use ODBC to read and write plain-text files. As long as you have the correct ODBC driver, Cold Fusion will support that data store.

Because Cold Fusion is an ODBC client, the database language used by Cold Fusion is SQL. To truly exploit the power of Cold Fusion, you must have a thorough understanding of SQL. Fortunately, by the end of this chapter, you should be enough of a SQL expert to start generating world-class Cold Fusion applications.

Creating an ODBC Data Source

ODBC client applications do not directly load ODBC drivers. In fact, they have no knowledge of what driver to use with any specific database. Rather, the application connects to a *data source*. A data source appears to your application as a virtual database. Within the data source, all the ODBC settings are configured, including specifying which ODBC driver to use.

Before your application can use an ODBC driver, you must create a data source. Doing so involves the following steps:

1. Select the ODBC driver that is appropriate for the database you plan to use. You have to install the driver if it is not already present on your computer.
2. Name your data source with a unique, and preferably descriptive, name.
3. Configure the driver-specific settings via the ODBC driver's configuration options.

After you create your data source, any ODBC client application can use it to access or manipulate the database with which it is associated.

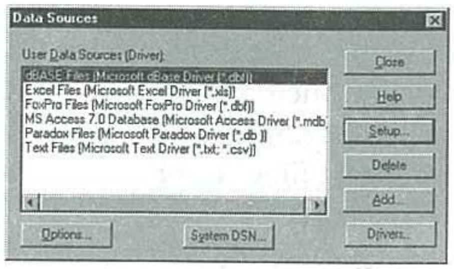
Note

The ODBC Control Panel applet and basic ODBC drivers are installed by many applications. If you have Microsoft Office installed, then you should have the applet and half dozen drivers installed, too.

If you need to obtain the applet, new ODBC drivers, or updated versions of existing drivers, the best place to start is the Microsoft FTP server at ftp.microsoft.com.

The ODBC Data Source Control Panel Applet

You configure ODBC data sources from within the ODBC applet in the Windows Control Panel. Try bringing up the Windows Control Panel. You should see an applet called *ODBC* or *32bit ODBC*. Double-click the ODBC applet to open the ODBC Data Sources dialog box, as shown in Figure 8.4.

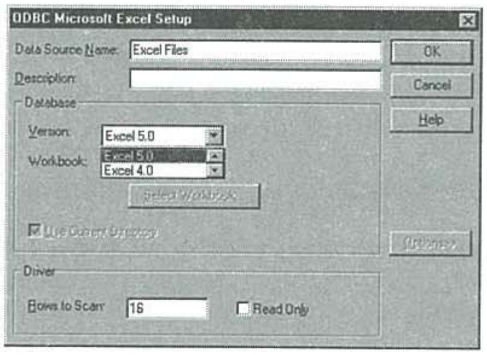


dBASE Files (Microsoft dBase Driver (*.dbf))
Excel Files (Microsoft Excel Driver (*.xls))
FoxPro Files (Microsoft FoxPro Driver (*.dbf))
MS Access 7.0 Database (Microsoft Access Driver (*.mdb
Paradox Files (Microsoft Paradox Driver (*.db))
Text Files (Microsoft Text Driver (*.txt; *.csv))

Figure 8.4 In the ODBC Data Sources dialog box, you can create and configure data sources and obtain driver version information.

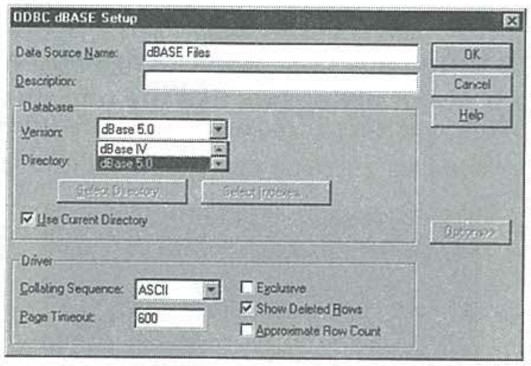
The User Data Sources box shows the currently installed ODBC data sources, including generic data sources for accessing Microsoft Excel, Microsoft FoxPro, and Text Files. Double-clicking any data source opens the ODBC Setup window for the driver associated with that data source.

Figures 8.5, 8.6, and 8.7 show the Microsoft Excel Setup window, the dBASE Setup window, and the Microsoft SQL Server Setup window, respectively. Each Setup window has a required Data Source Name field and an optional Description field. All other options are driver-specific, and, therefore, vary from one driver to the next.



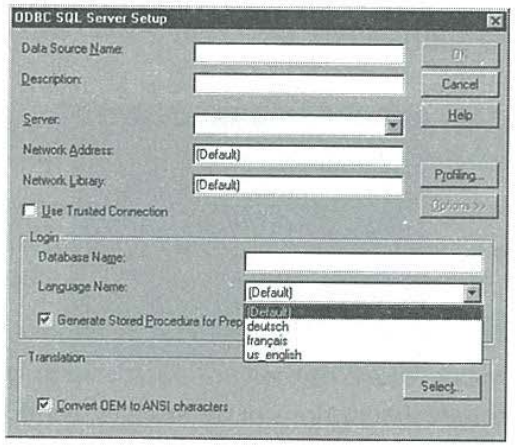
Data Source Name: Excel Files
Description:
Version: Excel 5.0
Workbook: Excel 5.0, Excel 4.0
☑ Use Current Directory
Rows to Scan: 16
☐ Read Only

Figure 8.5 The Microsoft Excel ODBC driver setup prompts for Excel-specific information, including the version of Excel and worksheet-related options.


ODBC dBASE Setup

Data Source Name: dBASE Files
Description:
Database
Version: dBase 5.0
dBase IV
dBase 5.0
Directory:
☑ Use Current Directory
Driver
Collating Sequence: ASCII
Page Timeout: 600
☐ Exclusive
☑ Show Deleted Rows
☐ Approximate Row Count

Figure 8.6 The dBASE ODBC driver setup prompts for dBASE-specific information, including the dBASE version.


ODBC SQL Server Setup

Data Source Name:
Description:
Server:
Network Address: (Default)
Network Library: (Default)
☐ Use Trusted Connection
Login
Database Name:
Language Name: (Default)
(Default)
deutsch
français
us_english
☑ Generate Stored Procedure for Prep
Translation
☑ Convert OEM to ANSI characters

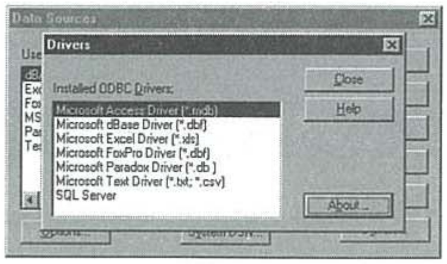
Figure 8.7 The Microsoft SQL Server ODBC driver setup prompts for network login and address information.

The ODBC Data Sources dialog box is also used to configure system wide ODBC options. These options are all accessed via the buttons listed in Table 8.2.

Table 8.2 ODBC Data Source Buttons

Button	Description
Options...	Configure system-wide ODBC options, such as tracing.
System DSN...	Set up data sources that the system, or any user, can use, rather than the local user.
Close	Close the ODBC Data Source dialog box.
Help	Obtain help.
Setup...	Configure the selected data source, which is the same as double-clicking a data source.
Delete	Permanently remove the selected data source.
Add...	Add a new data source using an existing ODBC driver.
Drivers...	Display a list of available ODBC drivers.

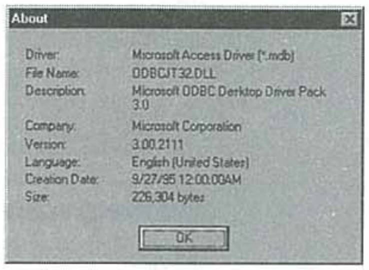
Try clicking the Drivers... button now. A Drivers dialog box similar to the one shown in Figure 8.8 should appear. In this dialog box, select any driver by clicking it, and then click the About... button. An About dialog box like the one shown in Figure 8.9 should then appear.



Installed ODBC Drivers:

```
Microsoft Access Driver (*.mdb)
Microsoft dBase Driver (*.dbf)
Microsoft Excel Driver (*.xls)
Microsoft FoxPro Driver (*.dbf)
Microsoft Paradox Driver (*.db )
Microsoft Text Driver (*.txt; *.csv)
SQL Server
```

Figure 8.8 You can check to see what ODBC drivers are installed on your system by clicking the Data Source dialog box's Drivers button.



```
Driver:          Microsoft Access Driver (*.mdb)
File Name:       ODBCJT32.DLL
Description:     Microsoft ODBC Desktop Driver Pack
                 3.0
Company:         Microsoft Corporation
Version:         3.00.2111
Language:        English (United States)
Creation Date:   9/27/95 12:00:00AM
Size:            226,304 bytes
```

Figure 8.9 ODBC drivers include descriptions of themselves, vendor information, the release date, and a version number.

> **Note**
>
> If you look at the About information for the Access, dBASE, Excel, FoxPro, Paradox, and Text drivers, you may notice that they are all in fact the same driver. Microsoft supplies all these drivers as part of its ODBC Desktop Driver Pack, and they are installed automatically with the ODBC applet.

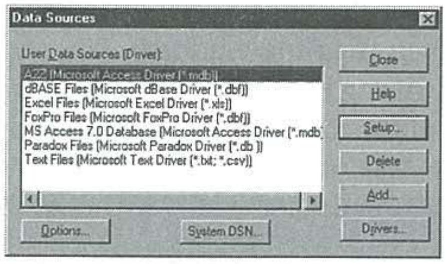
Creating a Data Source for the A2Z Books Database

Now that you've learned about data sources, you're ready to put all this newly acquired knowledge to use. In Chapter 8, "Creating Databases and Tables," you created a Microsoft Access database called A2Z. Now you're going to create an ODBC data source for this data file. Here are the steps:

1. Select the ODBC applet from the Windows Control Panel.
2. Click the Add... button to open the Add Data Source dialog box.
3. Select Microsoft Access Driver from the Installed ODBC Drivers list, and click OK to open the ODBC Microsoft Access Setup dialog box.
4. Name the data source by typing **A2Z** in the Data Source Name field.

5. Click the Select... button to locate the A2Z.MDB file.
6. Click OK to save your new data source.

That's all there is to it. The ODBC Data Sources dialog box now shows the new data source, A2Z, in the list of available User Data Sources, as shown in Figure 8.10.



A2Z (Microsoft Access Driver (*.mdb))
dBASE Files (Microsoft dBase Driver (*.dbf))
Excel Files (Microsoft Excel Driver (*.xls))
FoxPro Files (Microsoft FoxPro Driver (*.dbf))
MS Access 7.0 Database (Microsoft Access Driver (*.mdb
Paradox Files (Microsoft Paradox Driver (*.db))
Text Files (Microsoft Text Driver (*.txt; *.csv))

Figure 8.10 When you add new data sources, they appear in the list of available User Data Sources.

Note

Don't confuse ODBC data sources and ODBC drivers. ODBC drivers are dynamic link libraries, or DLLs, that communicate with a specific data store type. A data source is a complete database configuration that uses an ODBC driver to communicate with a specific database.

A data source communicates with only one database. To use an ODBC driver to communicate with two or more of the same types of databases, you need to create multiple data sources that all use the same ODBC driver.

Using Microsoft Query

Now that you have a data source, all you need is a client application with which to access the data. Ultimately, the client you will use is Cold Fusion—after all, that is why you're reading this book. But to start learning SQL without having to learn Cold Fusion, you need to start with Microsoft Query.

Microsoft Query is an SQL query utility. It is a simple ODBC database front end that Microsoft supplies with many of their other applications, including Microsoft Office. With Microsoft Query, you can test ODBC connectivity, interactively build SQL statements, and view the results of SQL queries, all in an easy-to-use environment. Microsoft Query is therefore a useful development and prototyping tool, and one well worth learning.

Note

If you set up Microsoft Office using the minimum setup, then you might not have Microsoft Query installed. If this is the case, run the Office setup program again and select Microsoft Query from the database tools option.

Tip

As you start developing Cold Fusion applications, you will find that most data-retrieval problems are in fact caused by incorrect SQL statements. Microsoft Query is a useful debugging tool because it enables you to test SQL statements interactively. Using Microsoft Query is a powerful way to validate SQL queries and to isolate data-retrieval problems.

Now run Microsoft Query. When the program loads, you should see a screen similar to the one shown in Figure 8.11. Along the top of the screen is the toolbar that gives you quick access to commonly used functions. The toolbar buttons are described in Table 8.3.

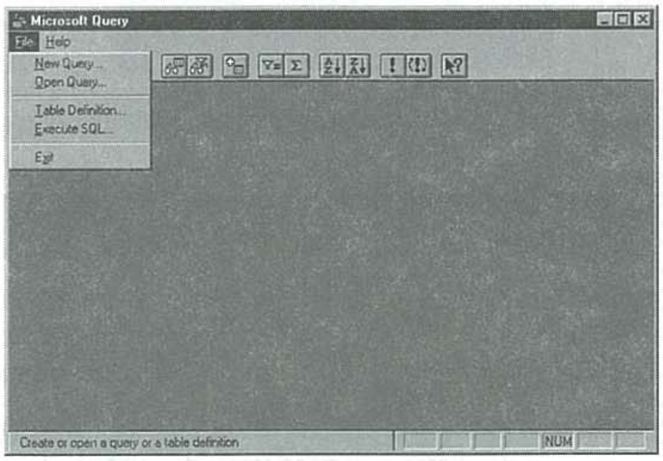


Figure 8.11 Microsoft Query is a multiple document interface (MDI) application. Using it, you can open multiple documents, or in this case queries, at once.

Table 8.3 The Microsoft Query Toolbar

Button	Effect
	Create a new query.
📂	Open a saved query.
💾	Save the currently selected query.
SQL	View or edit a query's SQL statement directly.
	Show or hide the available tables pane.
	Show or hide the selection criteria pane.

Button	Effect
	Include additional tables in the currently selected query.
∀=	Show only records that match the value of the selection.
Σ	Cycle through available for the currently selected column.
A Z ↓	Sort the table via the currently selected column in ascending order.
Z A ↓	Sort the table via the currently selected column in descending order.
!	Execute the query immediately.
(!)	Automatically execute the query as it is created and changed, and show results immediately.
?	Display Microsoft Query online help.

Preparing to Create Queries

You are now ready to create your first query. Click the New Query button to open the Select Data Source dialog box, as shown in Figure 8.12.

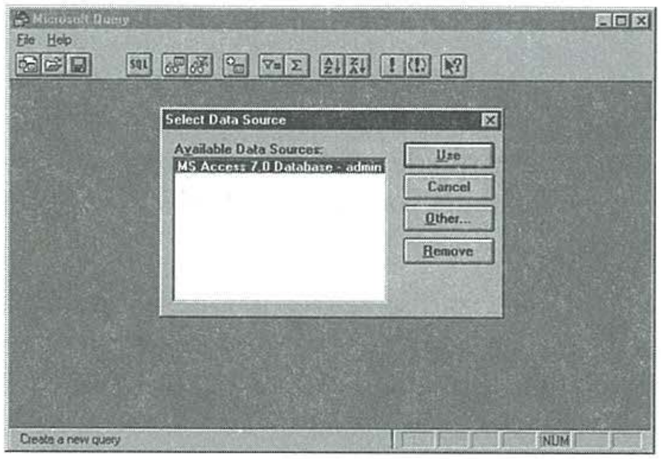


Figure 8.12 In the Microsoft Query Select Data Source dialog box, you can select the data source for your new query.

The first time Microsoft Query uses a data source you need to add it to the Select Data Source dialog box. To do so, click the Other... button to view the currently available data sources, then select the A2Z data source that you just created and click OK.

The A2Z data source then appears in the Select Data Source dialog box, as shown in Figure 8.13. At this point, select the A2Z data source, and then click Use.

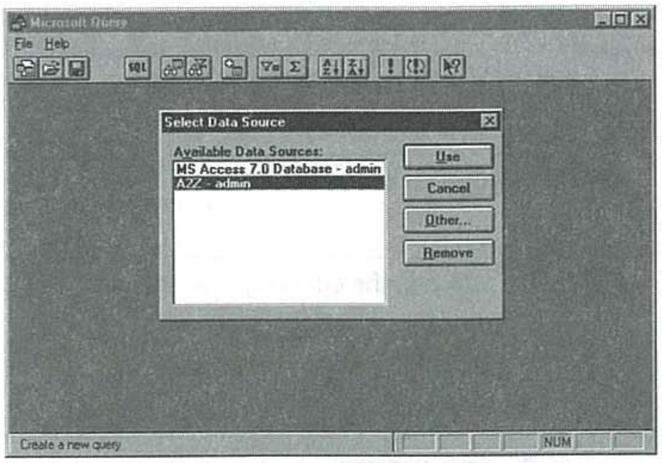


Figure 8.13 The first time Microsoft Query uses a data source you need to add it to the Select Data Source dialog box.

Creating Queries

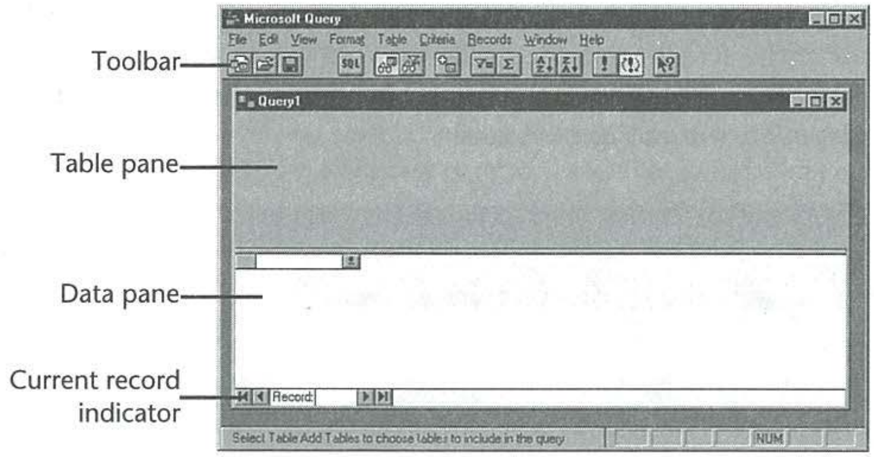
With all the preliminaries taken care of, you can roll up your sleeves and start writing SQL. The SQL statement that you will most use is the SELECT statement. You use SELECT, as its name implies, to select data from a table.

Most SELECT statements require at least the following two parameters:

- What data you want to select, known as the *select list*. If you specify more than one item, then you must separate each with a comma.
- The table (or tables) to select the data from, specified with the FROM keyword.

When you click Use in the Select Data Source dialog box to open a data source in a new query, Microsoft Query prompts you for the tables to include in this query. This feature is useful for interactively building queries. But because you're going to learn how to create queries by writing SQL statements yourself, don't select any tables now. Just click the Close button.

Once you have selected your data source, Microsoft Query will display the Query window, as shown in Figure 8.14. The top half is used by Microsoft Query to show tables in use and to display their relationships graphically if any are defined. In the bottom half of the screen, the results of your query are displayed.



Select Table Add Tables to choose tables to include in the query

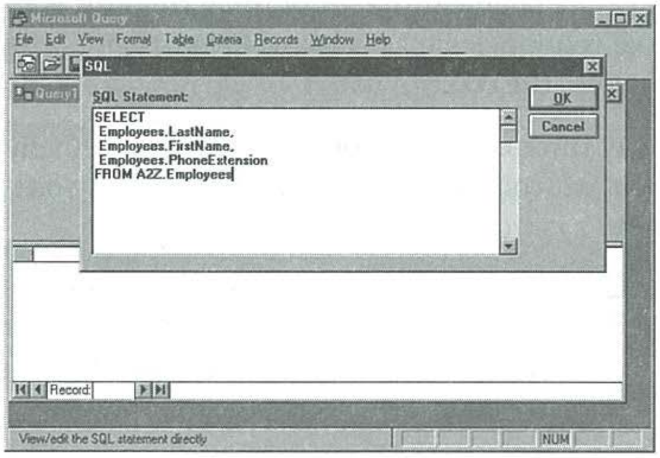
Figure 8.14 The Microsoft Query window is split into a table pane and a data pane.

Click the View SQL button (or choose SQL… from the View menu) to open the SQL window. Here, you can view the SQL statement that produced the query results shown, and you also can create and modify SQL statements directly.

The first SQL SELECT you will create is a query for a list of employees' last names and phone extensions. Type the code in Listing 8.1 into the SQL Statement box, as shown in Figure 8.15, and then click OK.

Listing 8.1 Simple SELECT Statement

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
```



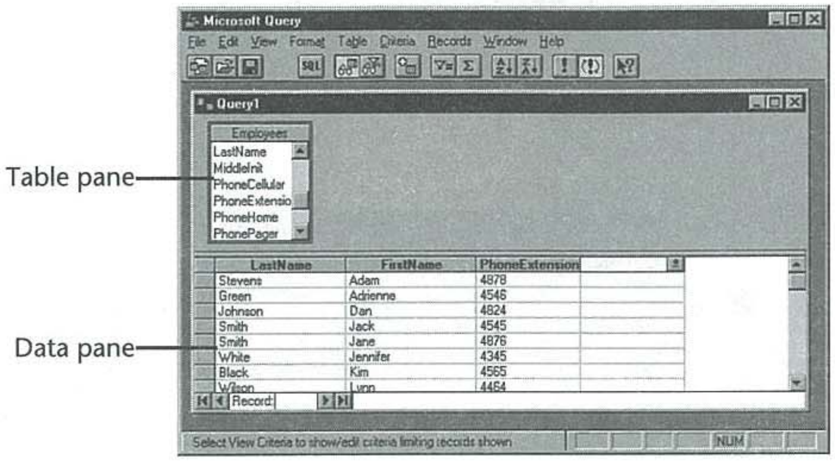
SQL Statement:

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
```

View/edit the SQL statement directly

Figure 8.15 In the SQL window, you can view generated SQL or enter SQL statements directly.

That's it! You've written your first SQL statement. Microsoft Query shows the table you are using in the top half of the screen, and the results of your query appear in the bottom half. You should have 10 records listed, the same 10 records you entered into Microsoft Access directly, as shown in Figure 8.16.



LastName	FirstName	PhoneExtension
Stevens	Adam	4878
Green	Adrienne	4546
Johnson	Dan	4824
Smith	Jack	4545
Smith	Jane	4876
White	Jennifer	4345
Black	Kim	4565
Wilson	Lynn	4464

Select View Criteria to show/edit criteria limiting records shown

Figure 8.16 Microsoft Query displays query results in the data pane, the bottom part of the Query window.

Note

You an enter SQL statements on one long line or break them up over multiple lines. All white-space characters (spaces, tabs, new-line characters) are ignored when the command is processed. If you break a statement onto multiple lines and indent parameters, you make the statement easier to read and debug.

Before going any further, take a closer look at the SQL code you entered. The first parameter you pass to the SELECT statement is a list of three columns you want to see. A column is specified as table.column, such as Employees.LastName, where Employees is the table name and LastName is the column name.

Because you want specify three columns, you have to separate them with commas. No comma appears after the last column name, so if you have only one column in your select list, you don't need a comma.

Right after the select list, you specify the table on which you want to perform the query. You always precede the table name with the keyword FROM. The table name itself is fully qualified, meaning it is specified as database.table, or in this case A2Z.Employees.

Note

SQL statements are not case-sensitive; that is, you can specify the SELECT statement as SELECT, select, Select, or however you want. Common practice, however, is to enter all SQL keywords in uppercase and parameters in lowercase or mixed case. This way, you can read the SQL code and spot typos more easily.

Now modify the SELECT statement so it looks like the code in Listing 8.2. Click the SQL button to make the code changes, and then click OK.

Listing 8.2 SELECT All Columns

```
SELECT
 Employees.*
FROM A2Z.Employees
```

This time, instead of specifying explicit columns to select, you use an asterisk. The asterisk is a special select list option that represents all columns. The data pane now shows every column in the table in the order in which they appear in the table itself.

Caution

Generally, you should not use an asterisk in the select list unless you really need every column. Each column you select requires its own processing, and retrieving unnecessary columns can dramatically affect the retrieval times as your tables get larger.

Sorting Query Results

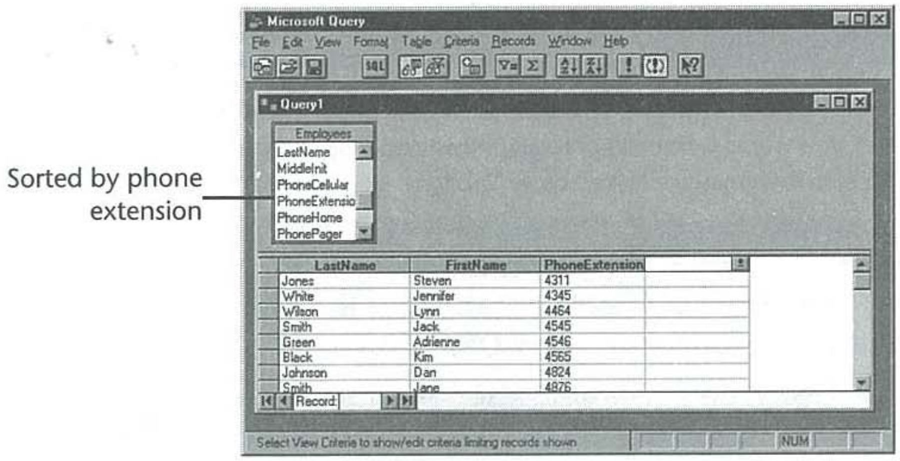
When you use the SELECT statement, the results are returned to you in the order in which they appear in the table. This is usually the order in which the rows were added to the table, typically not a sort order that is of much use to you. More often than not, when you retrieve data with a SELECT statement, you want to sort the query results. To sort rows, you need to add the ORDER BY clause. ORDER BY always comes after the table name; if you try to use it before, you generate an SQL error.

Now click the SQL button, and enter the SQL code shown in Listing 8.3. Then click OK.

Listing 8.3 SELECT with Sorted Output

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
ORDER BY PhoneExtension
```

Your output is then sorted by the PhoneExtension column, as shown in Figure 8.17.


Sorted by phone extension
Microsoft Query
File Edit View Format Table Criteria Records Window Help
Query1
Employees
LastName
MiddleInit
PhoneCellular
PhoneExtensio
PhoneHome
PhonePager
Record:
Select View Criteria to show/edit criteria limiting records shown
NUM

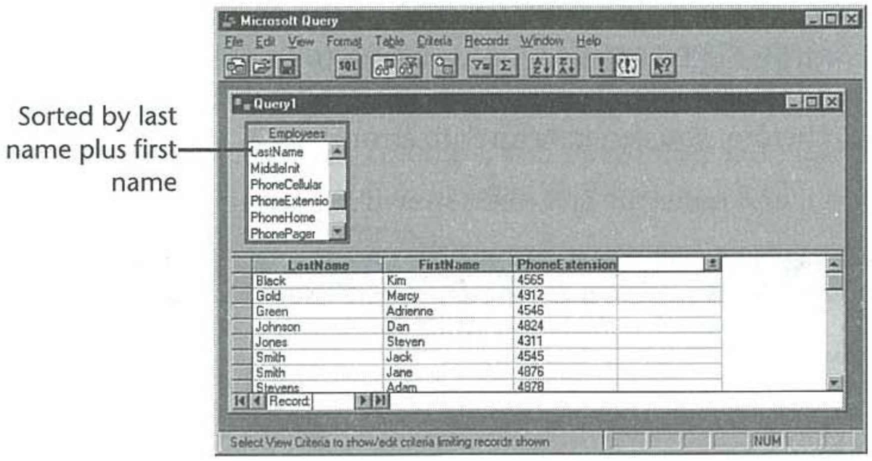
```
LastName   FirstName   PhoneExtension
Jones      Steven      4311
White      Jennifer    4345
Wilson     Lynn        4464
Smith      Jack        4545
Green      Adrienne    4546
Black      Kim         4565
Johnson    Dan         4824
Smith      Jane        4876
```

Figure 8.17 You use the ORDER BY clause to sort SELECT output.

What if you need to sort by more than one column, as you did in the beginning of Chapter 7? No problem. You can pass multiple columns to the ORDER BY clause. And once again, if you have multiple columns listed, you need to separate them with a comma. The SQL code in Listing 8.4 demonstrates how to sort on more than one column by sorting the employee list by last name plus first name. The sorted output is shown in Figure 8.18.

Listing 8.4 SELECT with Output Sorted on More than One Column

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
ORDER BY LastName, FirstName
```


Sorted by last name plus first name
Microsoft Query
File Edit View Format Table Criteria Records Window Help
Query1
Employees
LastName
MiddleInit
PhoneCellular
PhoneExtensio
PhoneHome
PhonePager
Record:
Select View Criteria to show/edit criteria limiting records shown
NUM

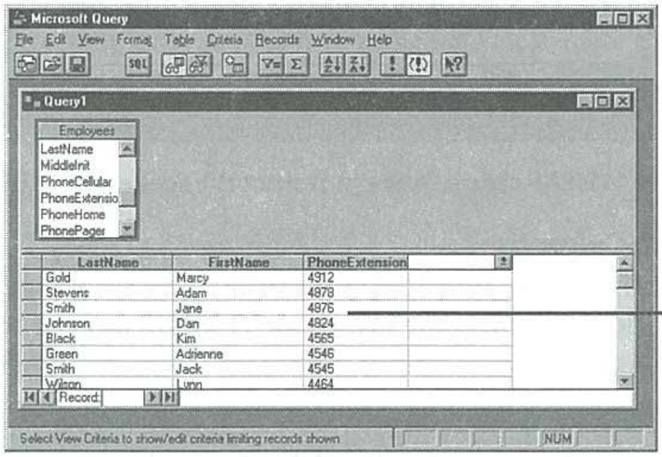
```
LastName   FirstName   PhoneExtension
Black      Kim         4565
Gold       Marcy       4912
Green      Adrienne    4546
Johnson    Dan         4824
Jones      Steven      4311
Smith      Jack        4545
Smith      Jane        4876
Stevens    Adam        4878
```

Figure 8.18 Using the ORDER BY clause, you can sort output by more than one column.

You also can use ORDER BY to sort data in descending order (from Z to A). To sort a column in descending order, just use the DESC (short for descending) parameter. Listing 8.5 retrieves all the employee records and sorts them by extension in reverse order. Figure 8.19 shows the output that this SQL SELECT statement generates.

Listing 8.5 SELECT with Output Sorted in Reverse Order

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
ORDER BY PhoneExtension DESC
```



LastName	FirstName	PhoneExtension
Gold	Marcy	4912
Stevens	Adam	4878
Smith	Jane	4876
Johnson	Dan	4824
Black	Kim	4565
Green	Adrienne	4546
Smith	Jack	4545
Wilson	Lynn	4464

Figure 8.19 Using the ORDER BY clause, you can sort data in a descending sort sequence.

Filtering Data

So far, all your queries have retrieved all the rows in the table. You also can use the SELECT statement to retrieve only data that matches a specific search criteria. To do so, you must use the WHERE clause and provide a restricting condition. If a WHERE clause is present, when the SQL SELECT statement is processed, every row is evaluated against the condition. Only rows that pass the restriction are selected.

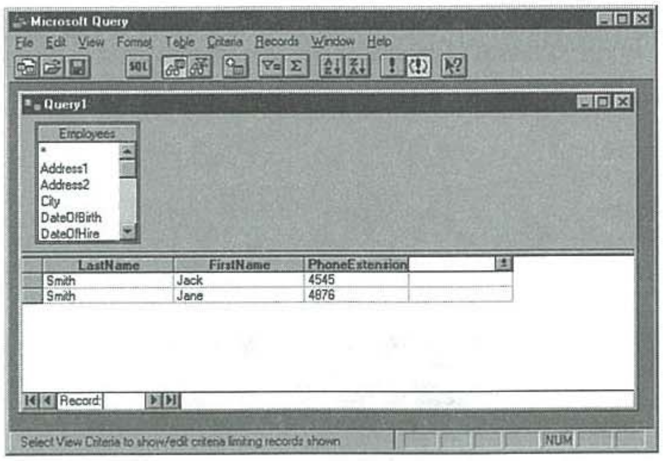
If you use a WHERE clause, it must appear after the table name. If you use both the ORDER BY and WHERE clauses, the WHERE clause must appear after the table name but before the ORDER BY.

Filtering on a Single Column

To demonstrate filtering, modify the SELECT statement to retrieve only employees whose last name is *Smith*. Listing 8.6 contains the SELECT statement, and the resulting output is shown in Figure 8.20.

Listing 8.6 SELECT with WHERE Clause

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
WHERE LastName = "Smith"
```



LastName	FirstName	PhoneExtension
Smith	Jack	4545
Smith	Jane	4876

Select View Criteria to show/edit criteria limiting records shown

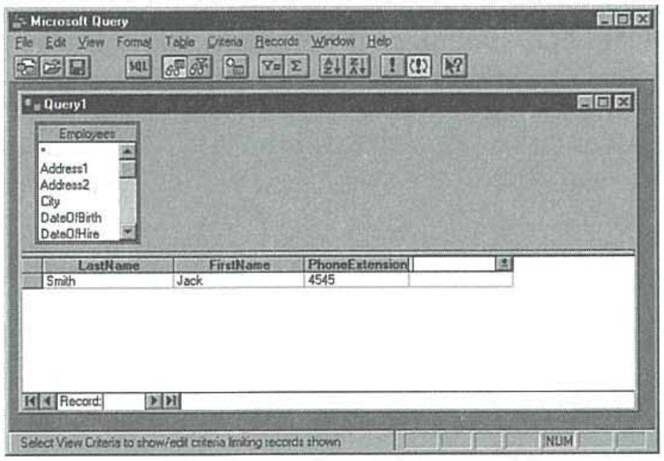
Figure 8.20 Using the WHERE clause, you can restrict the scope of a SELECT search.

Filtering on Multiple Columns

The WHERE clause also can take multiple conditions. To search for *Jack Smith*, you can specify a search condition in which the last name is *Smith* and the first name is *Jack*, as shown in Listing 8.7. As Figure 8.21 shows, only *Jack Smith* is retrieved.

Listing 8.7 SELECT with Multiple WHERE Clauses

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
WHERE LastName = "Smith" AND FirstName = "Jack"
```



LastName	FirstName	PhoneExtension
Smith	Jack	4545

Select View Criteria to show/edit criteria limiting records shown

Figure 8.21 Using multiple WHERE clauses, you can narrow down your search.

The AND and OR Operators

Multiple WHERE clauses can be evaluated as AND conditions or OR conditions. The example in Listing 8.7 is an AND condition. Only rows in which both the last name is

Smith and the first name is *Jack* will be retrieved. If you change the clause to the following, other employees with a last name of *Smith* are retrieved no matter what the first name:

WHERE LastName = "Smith" OR FirstName = "Jack"

Similarly, any employee named *Jack* is retrieved, regardless of the last name.

You can combine the AND and OR operators to create any search condition you need. Listing 8.8 and 8.9 show two different WHERE clauses that accomplish the exact same thing—specifically, retrieving only *Jack Smith* and *Kim Black*.

Listing 8.8 Combining WHERE Clauses with AND and OR Operators

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
WHERE (LastName = "Smith" AND FirstName = "Jack")
 OR (LastName = "Black" AND FirstName = "Kim")
```

Listing 8.9 Combining WHERE Clauses with AND and OR Operators

```
SELECT
 Employees.LastName,
 Employees.FirstName,
 Employees.PhoneExtension
FROM A2Z.Employees
WHERE (LastName = "Smith" OR LastName = "Black")
 AND (FirstName = "Jack" OR FirstName = "Kim")
```

Evaluation Precedence

When a WHERE clause is processed, the operators are evaluated in the following order of precedence:

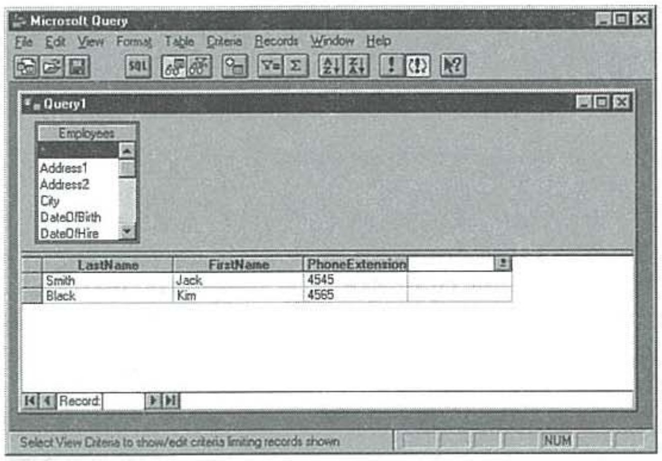
- Parentheses have the highest precedence.
- The AND operator has the next level of precedence.
- The OR operator has the lowest level of precedence.

What does this mean? Well, look at the WHERE clause in Listing 8.9. The clause reads WHERE (LastName = "Smith" OR LastName = "Black") AND (FirstName = "Jack" OR FirstName = "Kim"). This clause evaluates to the following:

(LastName = "Smith" OR LastName = "Black"). This clause retrieves only people whose last name is *Smith* or *Black*.

AND (FirstName = "Jack" OR FirstName = "Kim"). Of the names retrieved, this clause keeps only those whose first name is *Jack* or *Kim*. The rest are discarded.

The results of this query are shown in Figure 8.22. As you can see, only *Jack Smith* and *Kim Black* are retrieved, which is exactly the result you want.



LastName	FirstName	PhoneExtension
Smith	Jack	4545
Black	Kim	4565

Select View Criteria to show/edit criteria limiting records shown

Figure 8.22 With parentheses, you can control the precedence with which operators are evaluated.

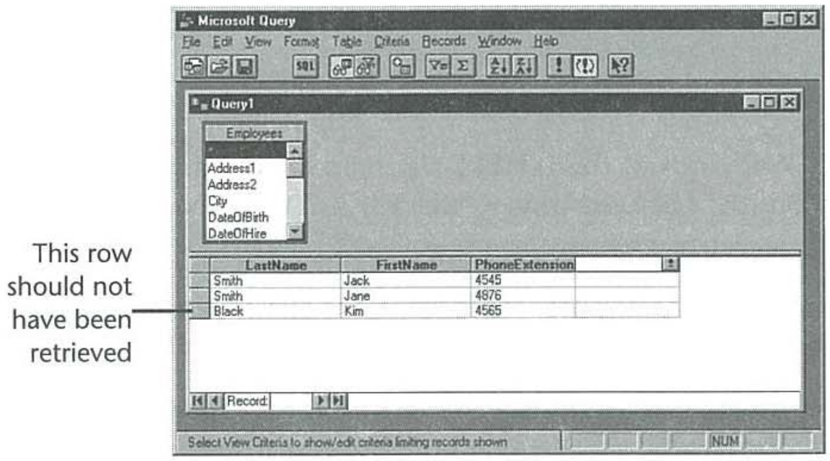
Without the parentheses, the clause would read WHERE LastName = "Smith" OR LastName = "Black" AND FirstName = "Jack" OR FirstName = "Kim". Because the AND operator takes precedence over the OR operator, this clause would be evaluated as follows:

WHERE LastName = "Smith". This clause retrieves anyone whose last name is *Smith*, regardless of first name.

OR LastName = "Black" AND FirstName = "Jack". This clause also retrieves anyone whose last name is *Black* and whose first name is *Jack*.

OR FirstName = "Kim". And finally, this clause also retrieves anyone whose first name is *Kim*.

The results of this query are shown in Figure 8.23. As you can see, *Jane Smith* is also retrieved. Because no parentheses bind the *Smith* restriction with the *Jack* restriction, the *Smith* restriction is evaluated by itself. *Jane Smith* is therefore a valid match.


This row should not have been retrieved
Microsoft Query
File Edit View Format Table Criteria Records Window Help
SQL
Query1
Employees
*
Address1
Address2
City
DateOfBirth
DateOfHire
Record:
NUM

LastName	FirstName	PhoneExtension
Smith	Jack	4545
Smith	Jane	4876
Black	Kim	4565

Select View Criteria to show/edit criteria limiting records shown

Figure 8.23 Without parentheses, the default order of precedence is used, and the results might not be what you expect.

Obviously, this result is not what you want. To force the correct evaluation precedence for your operators, you must use parentheses. This way, there is no doubt as to what you are trying to retrieve.

Tip

Always using parentheses whenever you have more than one WHERE clause is good practice. They make the SQL statement easier to read and easier to debug.

WHERE Conditions

For the examples to this point, you have used only the = (equal) operator. You filtered rows based on their being equal to a specific value. Many other operators and conditions can be used with the WHERE clause; they're listed in Table 8.4.

Table 8.4 WHERE Clause Search Conditions

Condition	Description
=	Equal to. Tests for equality
<>	Not equal to. Tests for nonequality.
<	Less than. Tests that the value on the left is less than the value on the right.
<=	Less than or equal to. Tests that the value on the left is less than or equal to the value on the right.
>	Greater than. Tests that the value on the left is greater than the value on the right.
>=	Greater than or equal to. Tests that the value on the left is greater than or equal to the value on the right.
BETWEEN	Tests that a value is in the range between two values; the range is inclusive.
EXISTS	Tests for the existence of rows returned by a subquery.
IN	Tests to see whether a value is contained within a list of values.
IS [NOT] NULL	Tests to see whether a column contains a NULL value (or a non-NULL value).
LIKE	Tests to see whether a value matches a specified pattern.
NOT	Negates any test.

= (Testing for Equality)

You use the = operator to test for value equality. The following example retrieves only employees whose last name is *Smith:*

```
WHERE LastName = "Smith"
```

< > (Testing for Nonequality)

You use the < > operator to test for value nonequality. The following example retrieves only employees whose first name is not *Kim:*

```
WHERE FirstName < > "Kim"
```

< (Testing for Less Than)

Using the < operator, you can test that the value on the left is less than the value on the right. The following example retrieves only employees whose last name is less than *C*, meaning that their last name begins with an *A* or a *B*:

```
WHERE LastName < “C”
```

<= (Testing for Less Than or Equal To)

Using the <= operator, you can test that the value on the left is less than or equal to the value on the right. The following example retrieves only employees whose phone extension is 4500 or less:

```
WHERE PhoneExtension < “4500”
```

> (Testing for Greater Than)

You use the > operator to test that the value on the left is greater than the value on the right. The following example retrieves only employees whose phone extension is greater than 4800:

```
WHERE LastName > “4800”
```

>= (Testing for Greater Than or Equal To)

You use the <= operator to test that the value on the left is greater than or equal to the value on the right. The following example retrieves only employees whose first name begins with the letter *J* or higher:

```
WHERE FirstName >= “J”
```

BETWEEN

Using the BETWEEN condition, you can to test whether a value falls into the range between two other values. The following example retrieves only employees whose phone extensions are between 4500 and 4600. Because the test is inclusive, extensions 4500 and 4600 are also retrieved.

```
WHERE PhoneExtension BETWEEN “4500” AND “4600”
```

The BETWEEN condition is actually nothing more than a convenient way of combining >= and <= conditions. You also could specify the preceding example as follows:

```
WHERE PhoneExtension >= “4500” AND PhoneExtension <= “4600”
```

The advantage of using the BETWEEN condition is that it makes the statement easier to read.

EXISTS

Using the EXISTS condition, you can check whether a subquery returns any rows. Subqueries are explained in Chapter 16, “Advanced SQL.”

IN

You can use the IN condition to test whether a value is part of a specific set. The set of values must be surrounded by parentheses and separated by commas. The following example retrieves employees whose last names are *Black*, *Jones*, or *Smith:*

```
WHERE LastName IN ("Black", "Jones", "Smith")
```

The preceding example is actually the same as the following:

```
WHERE LastName = "Black" OR LastName = "Jones" OR LastName = "Smith"
```

Using the IN condition does provide two advantages. First, it makes the statement easier to read. Second, and more important, you can use the IN condition to test whether a value is within the results of another SELECT statement. This issue is explained in Chapter 18.

IS [NOT] NULL

A NULL value is the value of a column that is empty. The IS NULL condition tests for rows that have a NULL value; that is, the rows have no value at all in the specified column. IS NOT NULL tests for rows that have a value in a specified column.

The following example retrieves all employees whose PhoneExtension is left empty:

```
WHERE PhoneExtension IS NULL
```

To retrieve only the employees who do have a phone extension, use the following example:

```
WHERE PhoneExtension IS NOT NULL
```

LIKE

Using the LIKE condition, you can test for string pattern matches using wild cards. Two wild-card types are supported. The % character means that anything from that position on is considered a match. You also can use [ ] to create a wild card for a specific character.

The following example retrieves employees whose last name begins with the letter *S*. To match the pattern, a last name must have an *S* as the first character and anything at all after it.

```
WHERE LastName LIKE "S%"
```

To retrieve employees with an *S* anywhere in their last names, you can use the following:

```
WHERE LastName LIKE "%S%"
```

You also can retrieve just employees whose last name ends with *S*, as follows:

```
WHERE LastName LIKE "%S"
```

The LIKE condition can be negated with the NOT operator. The following example retrieves only employees whose last name does not begin with *S:*

WHERE LastName NOT LIKE "S%"

Using the LIKE condition, you also can specify a wild card on a single character. If you want to find all employees named *Smith* but are not sure if the one you want spells his or her name *Smyth*, you can use the following:

WHERE LastName LIKE "Sm[iy]th"

This example retrieves only names that start with *Sm*, then have an *i* or *y*, and then a final *th*. With this example, as long as the first two characters are *Sm* and the last two are *th*, and as long as the middle character is *i* or *y*, the name is considered a match.

Tip

Using the powerful LIKE condition, you can retrieve data in many different ways. But everything comes with a price, and the price here is performance. Generally, LIKE conditions take far longer to process than other search conditions, especially if you use wild cards at the beginning of the pattern. As a rule, use LIKE and wild cards only when absolutely necessary.

From Here...

You covered a lot of ground in this chapter. You learned about SQL and ODBC and how to create ODBC data sources. You also learned how to use Microsoft Query to create and test SQL statements. The most used SQL statement is the SELECT statement, which you use to retrieve data from a table. Using the SELECT statement, you can specify from where the data should be retrieved, exactly what data to retrieve, and how to sort the resulting output.

For more information about topics mentioned in this chapter, see the following chapters:

- Chapter 6, "Database Fundamentals," teaches you what databases are and introduces you to important database terms and concepts.
- Chapter 7, "Creating Databases and Tables," teaches you how to take the design document created in this chapter and turn it into an actual set of database tables.
- Chapter 9, "SQL Data Manipulation," teaches you three other important SQL statements: the INSERT statement used to add new rows to a table, the UPDATE statement used to modify one or more rows in a table, and the DELETE statement used to delete one or more rows from a table.
- Chapter 10, "Cold Fusion Basics," teaches you how to create Cold Fusion code using the SQL you learned here.
- Chapter 16, "Advanced SQL," teaches the use of advanced SQL concepts, including views and table joins.

Chapter 10

Cold Fusion Basics

Ben Forta

In this chapter:

- **Understanding Cold Fusion templates**
- **Using Cold Fusion fields**
- **Creating data-driven templates**
- **Data drill-down**
- **Displaying results in tables**
- **Grouping results**

In Chapter 8, "Introduction to SQL," and Chapter 9, "SQL Data Manipulation," you learned the basics of SQL and interaction with SQL data sources. Now you're ready to start writing Cold Fusion applications. A Cold Fusion application is made up of one or more templates. A template is a file that contains HTML code as well as Cold Fusion Markup Language (CFML) code.

In this chapter, you do not learn how Cold Fusion works and what the various components that make up a Cold Fusion application are. That information is covered in detail in Chapter 2, "Introduction to Cold Fusion." In this chapter, you learn how to create a Cold Fusion template to present dynamic data output. You also learn different techniques for displaying and formatting data and how to implement a "drill-down" interface.

In this chapter, I introduce important Cold Fusion fundamentals. I encourage you to try every one of the examples here yourself because the lessons they demonstrate are the basis for everything covered in the remaining chapters.

Using Templates

As you learned in Chapter 2, all Cold Fusion interaction is via templates rather than HTML files. Templates can contain HTML, Cold Fusion tags and functions, or both.

Cold Fusion templates are plain-text files, just like HTML files are. But unlike HTML files, which are sent to the user's browser, templates are first processed by Cold Fusion. This way, you can embed instructions to Cold Fusion within your templates. If, for example, you want to process user-supplied parameters, retrieve data from a database, or conditionally display certain information, you can instruct Cold Fusion to do so.

But instead of just reading about templates, why don't you create one? The first template you will create just says "Hello" to you. Yes, I know that you can create the same response with any HTML file, but along with saying "Hello," this template also identifies your IP address and the browser you're using. You can't do that with plain HTML.

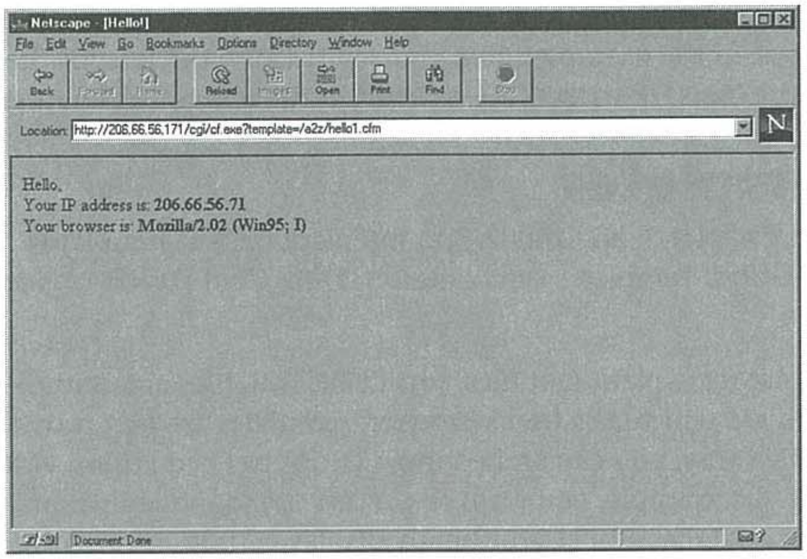
So, create a text file containing the code in Listing 10.1, and save it in your C:\A2Z\SCRIPTS directory as HELLO1.CFM.

On the CD

Listing 10.1 HELLO1.CFM—"Hello World" Cold Fusion Templates

```
<HTML>

<HEAD>
<TITLE>Hello!</TITLE>
</HEAD>

<BODY>

<CFOUTPUT>

 Hello,<BR>
 Your IP address is: <B>#REMOTE_ADDR#</B><BR>
 Your browser is: <B>#HTTP_USER_AGENT#</B><P>

</CFOUTPUT>

</BODY>

</HTML>
```

After you create and save the file, load your browser and type **http://*yourserver.com*/a2z/hello1.cfm** in the URL field (replacing *yourserver.com* with your own server name). Your browser should display a page that looks similar to the one shown in Figure 10.1. Of course, your IP address and browser information will be different.


Netscape - [Hello!]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/hello1.cfm
Document: Done

Hello,
Your IP address is: 206.66.56.71
Your browser is: Mozilla/2.02 (Win95; I)

Figure 10.1 Using Cold Fusion templates, you can display dynamic data in your Web pages.

Understanding Cold Fusion Templates

Now take a look at the code in Listing 10.1. Most of the code should be familiar to you as standard HTML. The tags for head, title, line breaks, and bold text are all the same HTML that you would use in any other Web page. What is not standard HTML is the `<CFOUTPUT>` tag and fields surrounded by pound signs (the # character).

All Cold Fusion-specific tags begin with CF; CFOUTPUT therefore is a Cold Fusion-specific tag. You use CFOUTPUT (or Cold Fusion Output) to mark a block of code that Cold Fusion should itself process prior to submitting it to the Web server for sending to your browser. When Cold Fusion encounters a `<CFOUTPUT>` tag, it scans all the text until the next `</CFOUTPUT>` for Cold Fusion functions or fields delimited by pound signs.

In Listing 10.1, you use two fields, `#REMOTE_ADDR#` and `#HTTP_USER_AGENT#`. They are CGI variables that the HTTP server makes available to CGI applications such as Cold Fusion. `#REMOTE_ADDR#` contains the IP address of your browser, and `#HTTP_USER_AGENT#` contains the string with which your browser identifies itself. When Cold Fusion encounters the text `#REMOTE_ADDR#` in the CFOUTPUT block, it replaces the text with the value in the REMOTE_ADDR CGI variable. And when Cold Fusion encounters `#HTTP_USER_AGENT#` on the next line, it replaces that text with the appropriate CGI variable, too. Instead of sending the text you entered back to your browser, Cold Fusion replaces the file names with the field values and sends those values back to you instead.

◀◀ See the "Understanding Cold Fusion Fundamentals" section in Chapter 2 for a detailed discussion of CGI applications and variables.

▶▶ See the CD for "CGI Environment Variables," for a complete list of all CGI variables and descriptions of each.

So why do you need the `<CFOUTPUT>` block? Well, take a look at what Cold Fusion would have done without it. Listing 10.2 contains a modified version of the code you used earlier; the output appears twice this time, once within a CFOUPUT block and once not.

Listing 10.2 HELLO2.CFM—Demonstration of the Use of CFOUTPUT

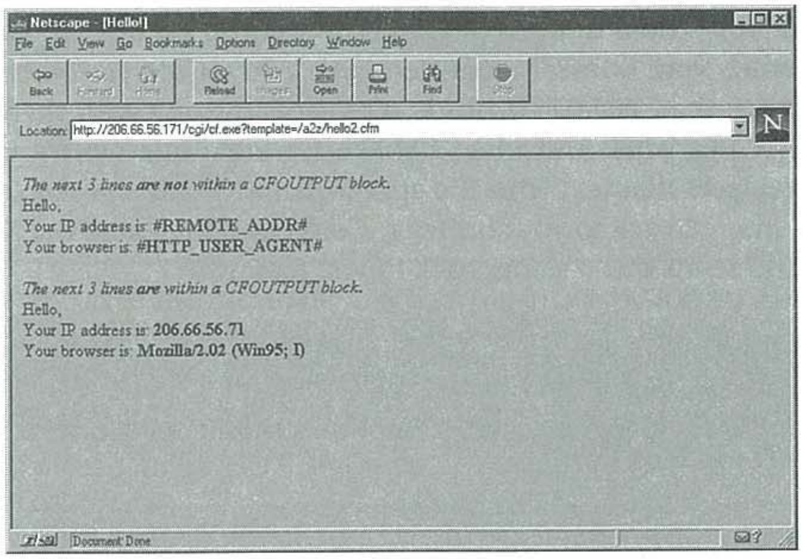
On the CD

```
<HTML>

<HEAD>
<TITLE>Hello!</TITLE>
</HEAD>

<BODY>

<I>The next 3 lines <B>are not</B> within a CFOUTPUT block.</I><BR>
Hello,<BR>
Your IP address is: <B>#REMOTE_ADDR#</B><BR>
Your browser is: <B>#HTTP_USER_AGENT#</B><P>
```

(continues)

Listing 10.2 Continued

```
<CFOUTPUT>

 <I>The next 3 lines <B>are</B> within a CFOUTPUT block.</I><BR>
 Hello,<BR>
 Your IP address is: <B>#REMOTE_ADDR#</B><BR>
 Your browser is: <B>#HTTP_USER_AGENT#</B><P>

</CFOUTPUT>

</BODY>

</HTML>
```


Netscape - [Hello!]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/hello2.cfm
Document: Done

*The next 3 lines **are not** within a CFOUTPUT block.*
Hello,
Your IP address is: **#REMOTE_ADDR#**
Your browser is: **#HTTP_USER_AGENT#**

*The next 3 lines **are** within a CFOUTPUT block.*
Hello,
Your IP address is: **206.66.56.71**
Your browser is: **Mozilla/2.02 (Win95; I)**

Figure 10.2 Fields not contained within a CFOUTPUT block are output as you enter them, not replaced with their values.

As you can see from the browser output in Figure 10.2, if you use fields outside a CFOUTPUT block, Cold Fusion displays the field name as you entered it, complete with the delimiting characters. More often than not, this result is not what you want.

Tip

Every `<CFOUTPUT>` must have a corresponding `</CFOUTPUT>` tag, and vice versa. If you omit either tag, Cold Fusion returns a syntax error.

Passing Parameters to Templates

In the first example, you used Cold Fusion to display dynamic data by specifying the field names for two CGI variables. You also can use Cold Fusion to display process parameters passed to a URL in exactly the same way.

To pass a parameter to a template, you could specify the parameter name and value within the URL. For example, to pass a parameter NAME with a value of BEN, you add `&NAME=BEN` to the URL. If you specify multiple URL parameters, then you must separate each one with an ampersand character (the & character).

Try this example yourself. Listing 10.3 contains a template that displays the value of a parameter called NAME, if it exists. To display the value, you use the `<CFIF>` tag to create a condition and a Cold Fusion function called ParameterExists. If the parameter NAME exists, then its value is displayed; otherwise, you are notified that the parameter is not passed.

▶▶ See the "Using Conditions" section in Chapter 12, "Using Forms to Add or Change Data," for a complete explanation of the CFIF tag and its usage.

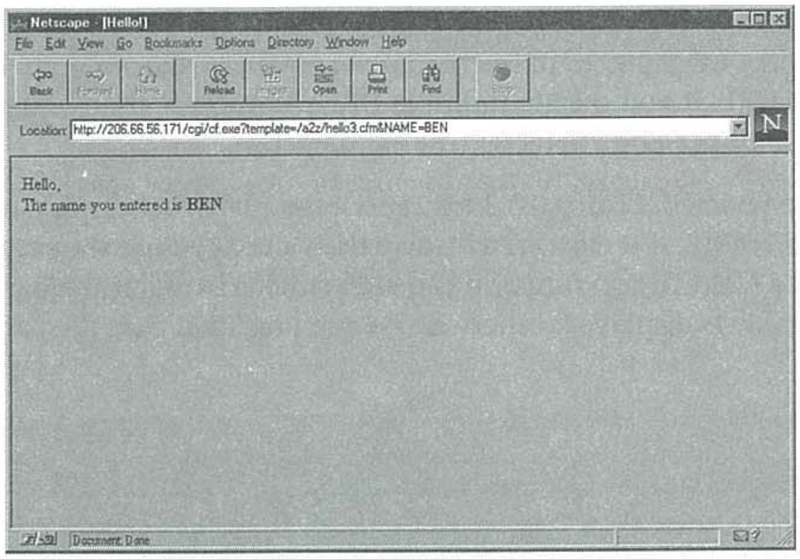
▶▶ See the "Using Optional Fields" section in Chapter 11, "Cold Fusion Forms," for an explanation of the ParameterExists function.

Listing 10.3 HELLO3.CFM—Demonstration of URL Parameter Processing

On the CD

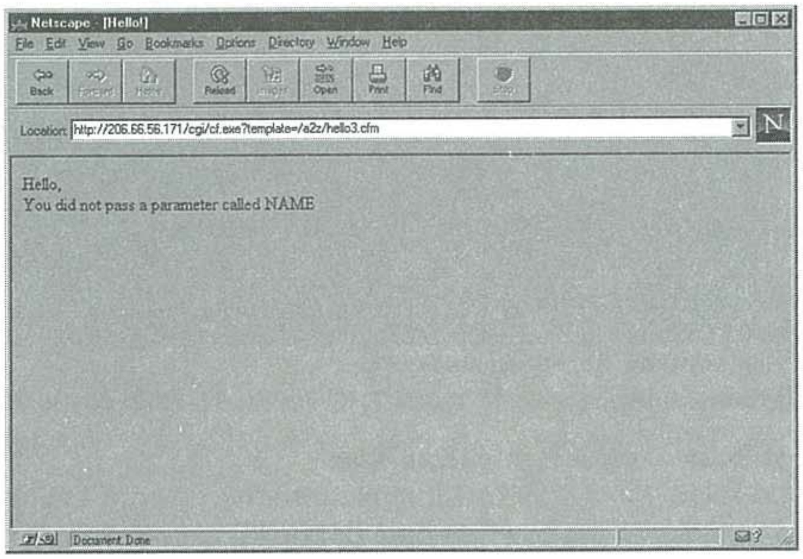
```
<HTML>

<HEAD>
<TITLE>Hello!</TITLE>
</HEAD>

<BODY>

Hello,<BR>

<CFIF #ParameterExists(name)# IS "Yes">
 <CFOUTPUT>
 The name you entered is <B>#name#</B>
 </CFOUTPUT>
<CFELSE>
 You did not pass a parameter called NAME
</CFIF>


</BODY>

</HTML>
```

After you create and save the file as HELLO3.CFM in the C:\A2Z\SCRIPTS directory, load your browser and type **http://*yourserver.com*/ a2z/hello3.cfm&NAME=BEN** (you don't have to use my name, any name will do). Your browser display should look like the one shown in Figure 10.3. Now try the example again, this time without any NAME parameter. You then should see a display like the one shown in Figure 10.4.


Netscape - [Hello!]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/hello3.cfm&NAME=BEN
Document: Done

Hello,
The name you entered is BEN

Figure 10.3 Cold Fusion converts parameters passed to a URL into fields that you can use within your template.


Netscape - [Hello!]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/hello3.cfm
Document: Done

Hello,
You did not pass a parameter called NAME

Figure 10.4 Whenever fields are optional, you should verify that they exist before using them.

So why go to the bother of testing for `#ParameterExists(name)#`? Well, try removing the `<CFIF>` statement (you have to remove the `<CFELSE>` and `</CFIF>` lines, too) and then type **http://yourserver.com/ a2z/hello3.cfm** without any NAME parameter. Cold Fusion returns an error message because it has no idea what `#name#` is. If you instruct Cold Fusion to process a field that does not exist, it complains.

Creating Data-Driven Templates

Now that you've seen what Cold Fusion templates look like and know how to create, save, and test them, return to the A2Z Books example.

Your employee database is set up and populated with data, so your next task is to publish this information on your intranet. This way, your users can access an up-to-date employee list at all times without needing any special software to do so. All they need to access the data is a Web browser.

Static Web Pages

Before you create a Cold Fusion template for your database, first take a look at how not to create this page. Listing 10.4 contains the HTML code for the employee list Web page. The HTML code is relatively simple; it contains header information and then a list of employees in an HTML unordered list `<UL>`.

Listing 10.4 EMPLOY.HTM—HTML Code For Employee List

On the CD

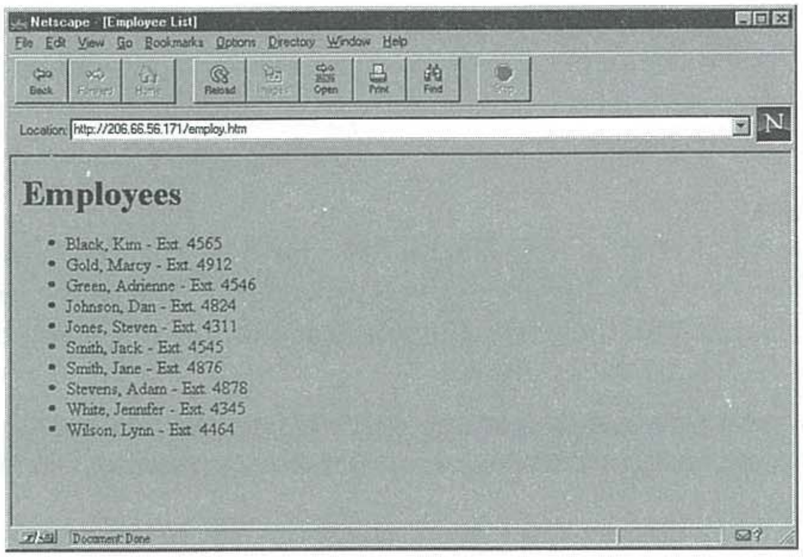
```
<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<UL>
 <LI>Black, Kim - Ext. 4565
 <LI>Gold, Marcy - Ext. 4912
 <LI>Green, Adrienne - Ext. 4546
 <LI>Johnson, Dan - Ext. 4824
 <LI>Jones, Steven - Ext. 4311
 <LI>Smith, Jack - Ext. 4545
 <LI>Smith, Jane - Ext. 4876
 <LI>Stevens, Adam - Ext. 4878
 <LI>White, Jennifer - Ext. 4345
 <LI>Wilson, Lynn - Ext. 4464
</UL>

</BODY>

</HTML>
```

Figure 10.5 shows the output that this code listing generates.

Dynamic Web Pages

Why, then, is a static HTML file not the way to create the Web page? Well, what would you do when a new employee is hired or when an employee leaves the company? What would you do if phone extensions change?

You could directly modify the HTML code to reflect these changes, but you already have all this information in a database. Why would you want to have to enter it all again? You would run the risk of making mistakes, misspelling names, getting entries out of order, and possibly even losing names altogether. And as the number of names in the list grows, so will the potential for errors occurring. Plus, during the period between updating the table and updating the Web page, employees will be looking at inaccurate information.


Netscape - [Employee List]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/employ.htm
Document: Done

Employees

- Black, Kim - Ext. 4565
- Gold, Marcy - Ext. 4912
- Green, Adrienne - Ext. 4546
- Johnson, Dan - Ext. 4824
- Jones, Steven - Ext. 4311
- Smith, Jack - Ext. 4545
- Smith, Jane - Ext. 4876
- Stevens, Adam - Ext. 4878
- White, Jennifer - Ext. 4345
- Wilson, Lynn - Ext. 4464

Figure 10.5 You can create the employee Web page as a static HTML file.

An easier and more reliable solution would be to have the Web page display the contents of your Employee table. This way, any table changes are immediately available to all employees. You can build the Web page dynamically based on the contents of the Employee table.

And so you create your first Cold Fusion template. Enter the code as it appears in Listing 10.5, and save it in the C:\A2Z\SCRIPTS as EMPLOY1.CFM. (Don't worry if the Cold Fusion code does not make much sense yet; I explain it in detail later in this chapter.)

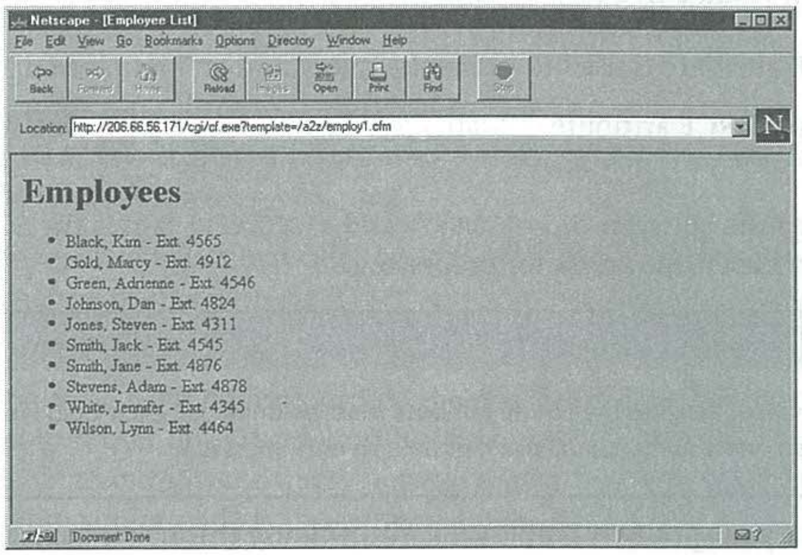
On the CD

Listing 10.5 EMPLOY1.CFM—Sample Cold Fusion Template

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employees"
>
SELECT FirstName, LastName, PhoneExtension
      FROM Employees
      ORDER BY LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<UL>

<CFOUTPUT QUERY="Employees">
 <LI>#LastName#, #FirstName# - Ext. #PhoneExtension#
</CFOUTPUT>
```

```
</UL>

</BODY>

</HTML>
```

Next, load your browser and type **http://*yourserver.com*/ a2z/employ1.cfm** in the URL field (again, replace *yourserver.com* with your own server name). The results are shown in Figure 10.6.


Netscape - [Employee List]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ1.cfm
Document: Done

Employees

- Black, Kim - Ext. 4565
- Gold, Marcy - Ext. 4912
- Green, Adrienne - Ext. 4546
- Johnson, Dan - Ext. 4824
- Jones, Steven - Ext. 4311
- Smith, Jack - Ext. 4545
- Smith, Jane - Ext. 4876
- Stevens, Adam - Ext. 4878
- White, Jennifer - Ext. 4345
- Wilson, Lynn - Ext. 4464

Figure 10.6 Ideally, the employee Web page should be generated dynamically, based on live data.

Now, compare Figure 10.5 and Figure 10.6. Can you see the difference between them? Look carefully.

Give up? Well, the truth is that they are not at all different. The screen shots are identical. If you were to look at the HTML source code that generated Figure 10.6, you would see that aside from lots of extra white space, the dynamically generated code is exactly the same as the static code you entered in Listing 10.4, and nothing like the dynamic code you entered in Listing 10.5.

Understanding Data-Driven Templates

So, how does the code in Listing 10.5 become the HTML source code that generates Figure 10.6? In the following sections, I help you review the code listing carefully.

The CFQUERY Tag

The first lines in Listing 10.5 are a Cold Fusion tag called CFQUERY. CFQUERY (or Cold Fusion Query) is the tag you use to submit any SQL statement to an ODBC data source. The SQL statement is usually an SQL SELECT statement but also can be INSERT, UPDATE, DELETE, or any other SQL statement.

◀◀ See the "Creating an ODBC Data Source" section in Chapter 9 for a more detailed discussion of ODBC data sources.

◀◀ See the "Creating Queries" section in Chapter 9 for a detailed discussion of SQL statements and specifically the SQL SELECT statement.

▶▶ See the "CFQUERY" section of Appendix A, "Cold Fusion Reference," for a detailed discussion of the CFQUERY tag, with examples showing the use of all attributes, because this chapter teaches how to use the CFQUERY tag but does not explain every attribute and feature.

The CFQUERY tag has several attributes, or parameters, that are passed to it when used. The CFQUERY in Listing 10.5 uses the following attributes:

- The NAME attribute is used to name the query and any returned data.
- The DATASOURCE attribute contains the name of the ODBC data source to be used.

Any text that appears between the `<CFQUERY>` and `</CFQUERY>` tags is the SQL code that will be passed to the ODBC driver for processing.

Note

The CFQUERY name passed to the NAME attribute must be unique in each Cold Fusion template. If you try to reuse a query name, Cold Fusion returns an error message.

The query NAME you specify is Employees. You will use this name later when you process the results generated by the query.

For the DATASOURCE attribute, you specify A2Z, the name of the data source you created in Chapter 9.

The SQL code we specified was:

```
SELECT FirstName, LastName, PhoneExtension FROM Employees ORDER BY LastName,
FirstName
```

This statement selects the columns you need from the Employee table and sorts them by last name plus first name. The SQL statement, like all other passed values, is enclosed within quotation marks.

Tip

The SQL statement in Listing 10.5 is broken up over many lines to make the code more readable. Although you can write a long SQL statement that is wider than the width of your browser window, generally you should break up these statements over as many lines as you need.

When Cold Fusion processes the template, the first item it finds is the Cold Fusion tag CFQUERY. Cold Fusion knows which tags it itself must process and which it must pass to the server directly. CFQUERY is a Cold Fusion tag and therefore must be processed by Cold Fusion.

When Cold Fusion encounters a CFQUERY tag, it creates an ODBC request and submits it to the specified data source. The results, if any, are stored in a temporary buffer and are identified by the name specified in the NAME attribute. This process happens before Cold Fusion processes the next line in the template.

The CFQUERY code, and indeed all Cold Fusion markup code, never gets sent on to the server for transmission to the browser. Unlike HTML tags that are browser instructions, CFML tags are instructions to Cold Fusion.

The next lines in the template are standard HTML tags: headers, title, and headings. Because they are not Cold Fusion tags, they are sent to the Web server and then to the client browser.

Displaying Query Results with the CFOUTPUT Tag

Next, in Listing 10.5, you create an HTML unordered list using the `<UL>` tag. The list is terminated a few lines later with a `</UL>` tag.

The list of employees itself goes between the `<UL>` and `</UL>` tags. Each name is a separate list item and therefore begins with an HTML `<LI>` tag. But instead of listing the employees as shown in Figure 10.5, you use a CFOUTPUT tag.

CFOUTPUT is the same Cold Fusion output tag you used earlier. But this time you use it to create a code block that outputs the results of a CFQUERY. For Cold Fusion to know which query results to output, the query name is passed to CFOUTPUT in the QUERY attribute. The name you provide is the same name assigned to the NAME attribute of the CFQUERY tag. In this case, the NAME is Employees.

The code between the `<CFOUTPUT QUERY="Employees">` and `</CFOUTPUT>` is the output code block. Cold Fusion uses this code once for every row that is retrieved. As 10 rows currently appear in the Employee table, the CFOUPUT code is looped through 10 times. And any HTML or CFML tags within that block are repeated as well, once for each row.

▶▶ See the "CFOUTPUT" section of Appendix A for a detailed discussion of the CFOUTPUT tag, with examples showing the use of all attributes, because this chapter explains how to use the CFOUTPUT tag and introduces only the features needed for the examples presented here.

Using Database Table Columns

As I explained earlier, Cold Fusion uses # to delimit fields. In addition to CGI variables and URL parameters, which you used at the beginning of this chapter, Cold Fusion fields can also be columns retrieved by a CFQUERY. Whatever field you use, Cold Fusion replaces the field name with the actual value. So when Cold Fusion processes the output block, it replaces `#LastName#` with the contents of the LastName column retrieved in the Employee query. Each time the output code block is used, that row's LastName value is inserted into the HTML code.

Cold Fusion fields can be treated as any other text in an HTML document. You can apply any of the HTML formatting tags to them. In the example, the query results need to be displayed in an unordered list. Each employee's name and phone extension is a list item and, therefore, is preceded by the `<LI>` tag. As the `<LI>` tag is included within the CFOUTPUT block, Cold Fusion outputs it along with every row.

So, for employee Kim Black at extension 4565, the line

```
<LI> #LastName#, #FirstName# - Ext. #PhoneExtension#
```

becomes

```
<LI> Black, Kim - Ext. 4565
```

Only the `<LI>` tag is within the CFOUPUT block, and not the `<UL>` and `</UL>`, because you want only one list, not many. If the `<UL>` and `</UL>` are within the CFOUPUT block, you have a new list created for each employee—definitely not the desired result at all.

Figure 10.6 shows the browser display that this template creates. It is exactly the same result as Figure 10.5, but without any new data entry whatsoever.

Welcome to Cold Fusion and the wonderful world of dynamic data-driven Web pages!

Using Drill-Down Applications

The nature of the World Wide Web places certain restrictions on data interaction. Every time a Web browser makes a request, a connection is made to a Web server, and that connection is maintained only for as long as it takes to retrieve the Web page. Subsequent selections and Web requests create yet another connection—again, just for the specific request.

Simple user interfaces that you may take for granted in most commercial software, such as scrolling through previous or next records with the cursor keys, become quite complex within the constraints of Web pages and how they interact with Web servers.

One elegant and popular form of Web-based data interaction is the "drill-down" approach. Drill down is designed to break up data so that only what is needed on a single page is displayed. Selecting an item in that page causes details about that item to be displayed. The processes is called *drilling down* because you drill through the data layer by layer to find the information you need.

The employee page you just created, for example, displays a simple list of employees and extensions. What if you want to display more information such as title, department, and e-mail address? You can just select more columns in the CFQUERY and display them in the CFOUTPUT code, but doing so would clutter the screen, making it hard to use. A better approach would be to display less information on a page and allow the user to click an employee's name to display more information about that employee. This approach, gradually digging deeper into a data set to find the information you want, is known as drilling down.

Building Dynamic SQL Statements

Creating a drill-down application in Cold Fusion involves creating multiple templates. For example, one template should list the employees, and a second template should display an employee's details.

First, create the detail template. The SQL query in this template has to select detailed user information for a specific user. Obviously, you don't want to create a template for every employee in your database. Doing so would totally defeat the purpose of using templates in the first place. Rather, the template needs to be passed a parameter, a value that uniquely identifies an employee. Fortunately, when you created the Employee table, you created a column called EmployeeID, which contains a unique employee ID for each employee in the table. The code in Listing 10.6 demonstrates how to pass parameters.

Listing 10.6 EMPDTL1.CFM—Passing Dynamic Parameters

On the CD

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employee"
>
SELECT LastName,
             FirstName,
             MiddleInit,
             Title,
             PhoneExtension,
             PhoneCellular,
             PhonePager,
             EMail
     FROM Employees
     WHERE EmployeeID = #EmployeeID#
</CFQUERY>

<CFOUTPUT QUERY="Employee">

 <HTML>

 <HEAD>
 <TITLE>#LastName#, #FirstName# #MiddleInit#</TITLE>
 </HEAD>

 <BODY>

 <H1>#LastName#, #FirstName#</H1>

 <HR>

 Title: #Title#
 <BR>
 Extension: #PhoneExtension#
 <BR>
 Cellular: #PhoneCellular#
 <BR>
 Pager: #PhonePager#
```

(continues)

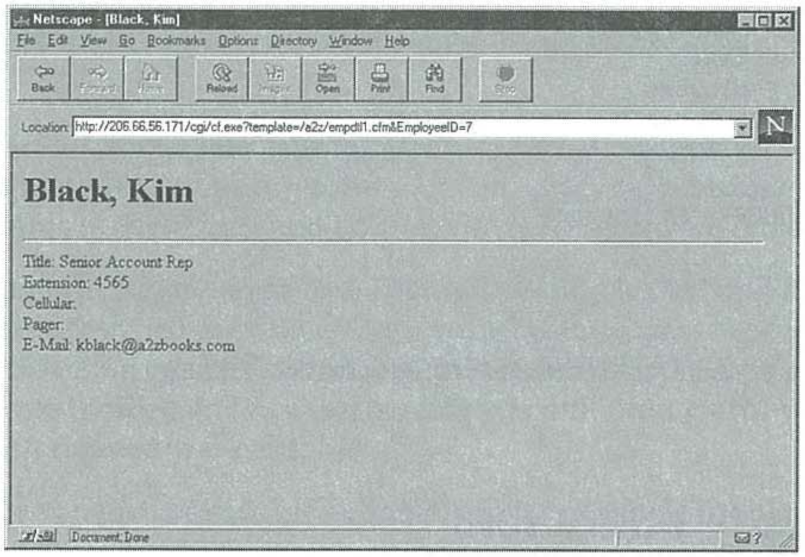
Listing 10.6 Continued

```
    <BR>
    E-Mail: #EMail#

    </BODY>

   </CFOUTPUT>
```

Before you look at the Web page produced by this code, take a look at the SQL statement in this CFQUERY tag. The SQL SELECT statement selects the columns needed and uses a WHERE clause to specify which row to select. The WHERE clause cannot be hard-coded for any particular employee ID and therefore uses a passed field, `#EmployeeID#`. The `#EmployeeID#` field is passed to the template as part of the URL.

If an EmployeeID of 7 is passed with the URL, therefore, the WHERE clause `WHERE EmployeeID = #EmployeeID#` becomes `WHERE EmployeeID = 7`—exactly what you need to select the correct row. As you learned earlier, parameters are passed to URLs after the template name, and each parameter is separated by an ampersand character. So, to specify employee ID 7, you add `&EmployeeID=7` to the URL.

Now try this example. Type the URL **http://*yourserver.com*/ a2z/ empdtl1.cfm?EmployeeID=7** in the URL field (once again, replace *yourserver.com* with your own server name) in your browser. The resulting output is shown in Figure 10.7.


Netscape - [Black, Kim]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=7
Document: Done

Black, Kim

Title: Senior Account Rep
Extension: 4565
Cellular:
Pager:
E-Mail: kblack@a2zbooks.com

Figure 10.7 If you want to create truly dynamic pages, parameters can be passed to Cold Fusion templates and used to create dynamic SQL statements.

To display the details for another employee, you just need to change the value passed to the URL EmployeeID parameter. Try replacing `EmployeeID=7` with `EmployeeID=5`. Changing the parameter displays information on a different employee. You can now use the same template to display details for any employee in the database because the Web page is data driven.

Implementing Data Drill Down

To complete the drill-down application, you need to modify the employee list page to include links to the employee details page. The code for the updated template is shown in Listing 10.7.

Listing 10.7 EMPLOY2.CFM—Building Dynamic SQL Statements

On the CD

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employees"
>
SELECT FirstName, LastName, PhoneExtension, EmployeeID
      FROM Employees
      ORDER BY LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<UL>

<CFOUTPUT QUERY="Employees">
 <LI><A HREF="empdtl1.cfm?EmployeeID=#EmployeeID#">
     #LastName#, #FirstName#</A> - Ext. #PhoneExtension#
</CFOUTPUT>

</UL>

</BODY>

</HTML>
```

Listing 10.7 is the same as Listing 10.5, with two exceptions. First, you now need the EmployeeID value, so you change the SQL SELECT statement in the CFQUERY to also include this column. Second, you modify the employee's name in the CFOUTPUT code block so that it is a hyperlink to the employee detail page.

The new employee name code reads

```
<LI> <A HREF="empdtl1.cfm?EmployeeID=#EmployeeID#">#LastName#, #FirstName#</
A> - Ext. #PhoneExtension#
```

When Cold Fusion processes employee 7, this line becomes

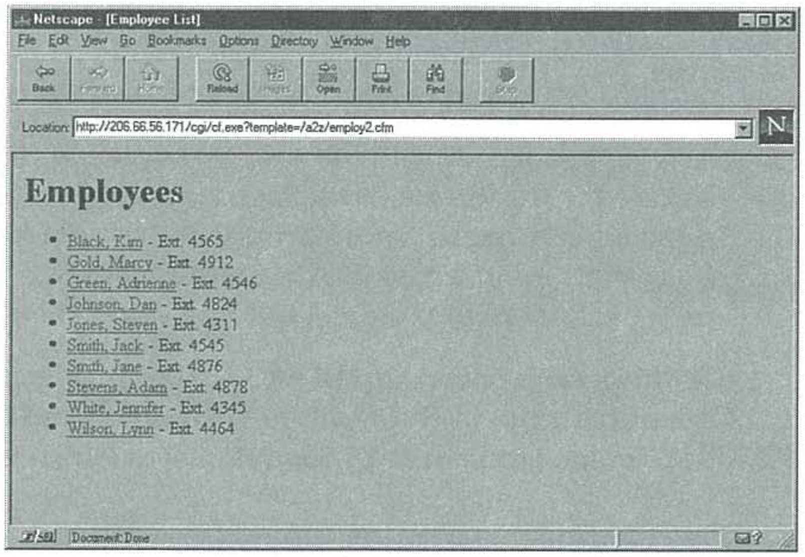
```
<LI> <A HREF="empdtl1.cfm?EmployeeID=7">Black, Kim</A> - Ext. 4565
```

III

Cold Fusion

This way, the URL needed for the hyperlink is dynamic, too. The URL built for each employee also contains the correct employee ID, which can be passed to the employee detail template.

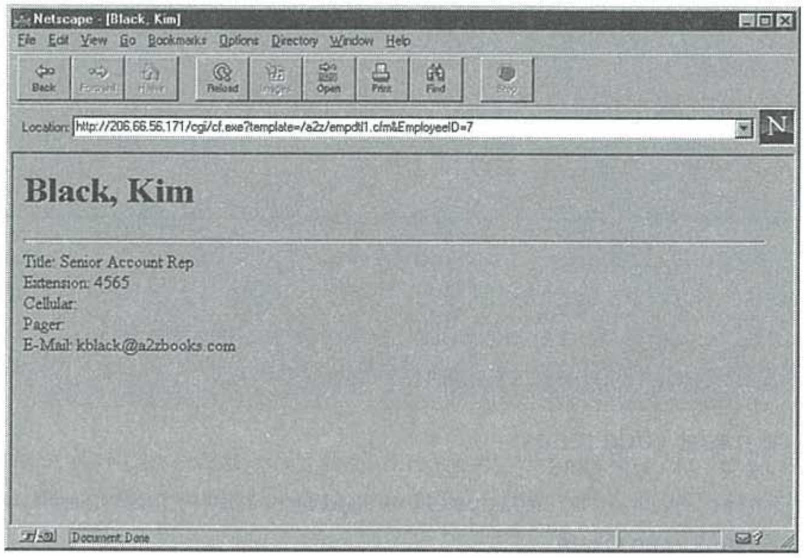
So now try out this example. Go to URL **http://*yourserver.com*/ a2z/employ2.cfm** (again, replace *yourserver.com* with your own server name). Figure 10.8 shows what the output looks like. The only difference between this display and the one in Figure 10.6 is that now the employee names are hyperlinks. You can click any one of these links to display employee details, as shown in Figure 10.9.


Netscape - [Employee List]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ2.cfm
Document: Done

Employees

- Black, Kim - Ext. 4565
- Gold, Marcy - Ext. 4912
- Green, Adrienne - Ext. 4546
- Johnson, Dan - Ext. 4824
- Jones, Steven - Ext. 4311
- Smith, Jack - Ext. 4545
- Smith, Jane - Ext. 4876
- Stevens, Adam - Ext. 4878
- White, Jennifer - Ext. 4345
- Wilson, Lynn - Ext. 4464

Figure 10.8 You can build hyperlink URLs dynamically to create even more dynamic Web pages.


Netscape - [Black, Kim]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=7
Document: Done

Black, Kim

Title: Senior Account Rep
Extension: 4565
Cellular:
Pager:
E-Mail: kblack@a2zbooks.com

Figure 10.9 By passing parameters to a Cold Fusion template, you can use the same template to display different records without requiring a different HTML page for each.

Using Frames to Implement Data Drill Down

One problem with the drill-down templates you just created is that every time you view an employee's details you have to click your browser's Back button to return to the employee list page. A more usable approach would be to display the employee list and details at the same time. Fortunately, you can do so easily by using a browser feature called *frames*. Using frames, you can split your browser window in two or more windows and control what gets displayed within each. And Cold Fusion templates are well suited for use within frames.

Creating frames involves creating multiple templates (or HTML pages). Each window in a frame typically displays a different template. If you have two windows, you need two templates. In addition, you always need one more page that is used to lay out and create the frames.

When you create the frames, each window is named with a unique name. In a nonframed window, every time you select a hyperlink, the new page is opened in the same window, replacing whatever contents were there previously. In a framed window, you can use the window name to control the destination for any output.

Creating Frames for Use with Cold Fusion

So, now that you have an idea how frames work, the first thing you need to do is create the template to define and create the frames. The code for template EMPLFRAM.CFM is shown in Listing 10.8.

Listing 10.8 EMPLFRAM.CFM—Employee Frame Definition and Creation

On the CD

```
<HTML>

<HEAD>
<TITLE>Employees</TITLE>
</HEAD>

<FRAMESET COLS="50%,50%">
 <FRAME SRC="/cgi/cf.exe?template=employ3.cfm" NAME="employees">
 <FRAME SRC="/cgi/cf.exe?template=empdtl1.cfm?EmployeeID=0" NAME="details">
</FRAMESET>

</HTML>
```

This template first defines the frames. `<FRAMESET COLS="50%,50%">` creates two columns (or windows), each taking up 50 percent of the width of the browser window.

Then the two columns are defined. The line `<FRAME SRC="employ3.cfm"NAME="employees">` creates the left frame. The NAME attribute names the window, and the SRC attribute specifies the name of the template to initially display within the window when the frame is first displayed.

When the frame is first displayed, no employee is selected yet. Therefore, no information is available for display in the details window, the right frame. The simplest way to display an empty frame is to specify an inexistent EmployeeID in the URL. We specified an

EmployeeID of 0, and so Cold Fusion finds no rows and does not display anything there at all.

Now, the next thing to do is to create the employee list template. Actually, it is the same as the one in Listing 10.7, with one important difference. The URL to display the employee details must include a TARGET attribute to designate which window to display the URL in. If the TARGET is omitted, the new data is displayed in the frame from which it is selected.

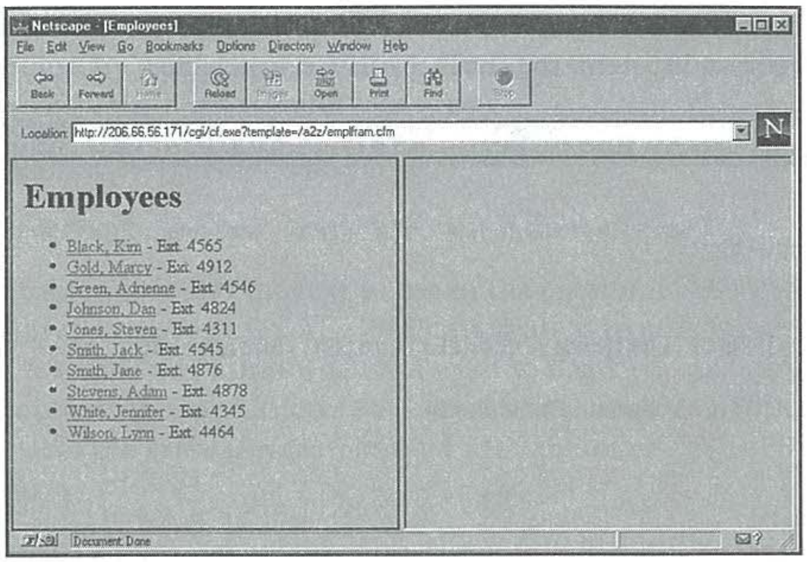
The modified code is shown in Listing 10.9. As you can see, the URL has been modified to include the attribute `TARGET="details"`. This attribute specifies that the new URL should be displayed in the frame named details, the right window.

On the CD

Listing 10.9 EMPLOY3.CFM—Using Frames for Data Drill Down

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employees"
>
SELECT FirstName, LastName, PhoneExtension, EmployeeID
      FROM Employees
      ORDER BY LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<UL>

<CFOUTPUT QUERY="Employees">
 <LI><A HREF="empdtl1.cfm?EmployeeID=#EmployeeID#"
     TARGET="details">#LastName#, #FirstName#</A> - Ext. #PhoneExtension#
</CFOUTPUT>

</UL>

</BODY>

</HTML>
```

That's all there is to it. To try out this example, go to URL **http://*yourserver.com*/a2z/emplfram.cfm**. (I am no longer going to remind you to replace *yourserver.com* with your own server name.) Figure 10.10 shows the output as it appears in framed windows. Try clicking any employee's name in the left window, and the right window then displays employee details.


Netscape - [Employees]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location:

http://206.66.56.171/cgi/cf.exe?template=/a2z/emplfram.cfm

Employees

- Black, Kim - Ext. 4565
- Gold, Marcy - Ext. 4912
- Green, Adrienne - Ext. 4546
- Johnson, Dan - Ext. 4824
- Jones, Steven - Ext. 4311
- Smith, Jack - Ext. 4545
- Smith, Jane - Ext. 4876
- Stevens, Adam - Ext. 4878
- White, Jennifer - Ext. 4345
- Wilson, Lynn - Ext. 4464

Document: Done

Figure 10.10 Cold Fusion is well suited for use within HTML frames.

Displaying Results in Tables

Most Web browsers now support tables. By using the HTML <TABLE> tag, you can display data in a two-dimensional grid. Tables are useful for presenting lists in a clean, columnar display.

Because HTML tables are used so often to display query results in data-driven pages, and the <TABLE> syntax can be confusing at times, the makers of Cold Fusion created a Cold Fusion tag called CFTABLE. The CFTABLE tag is designed to conceal the details involved in creating HTML tables. All you have to do is tell Cold Fusion what data to put in each column, and Cold Fusion generates the <TABLE> markup code for you.

The CFTABLE tag has another important advantage. It enables you to create tables that can be viewed by all browsers, even those that do not support HTML tables. To do this, Cold Fusion renders the output in a nonproportional font and pads fields with spaces so that they line up in columns. Although the resulting table might not look as good as a true HTML table, it is functional and is supported by all browsers.

▶▶ See the "CFTABLE" section in Appendix A for a detailed discussion of the CFTABLE tag, with examples showing the use of all attributes and how it can be used. This chapter teaches the basic use of the CFTABLE tag and introduces only the features needed for the examples presented here.

Creating Non-HTML Tables with CFTABLE

For an example of the places where you can use CFTABLE, look at the browser output shown in Figure 10.8. Notice how the phone extension is right next to the name and in a different location on the screen depending on how long the employee's name is. If the employees were listed in a table, the data could be presented in a cleaner and more organized fashion.

Listing 10.10 is based on Listing 10.7, but instead of using an unordered list and presenting each employee as a list item, the list is displayed in a table.

On the CD

Listing 10.10 EMPLOY4.CFM—Using CFTABLE to Display Data in Tables

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employees"
>
SELECT FirstName, LastName, PhoneExtension, EmployeeID
      FROM Employees
      ORDER BY LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<
 CFTABLE
 QUERY="Employees"
 COLHEADERS
>
 <
  CFCOL
  HEADER="Employee"
  TEXT="<A HREF=""empdtl1.cfm?EmployeeID=#EmployeeID#"">#LastName#,
#FirstName#</A>"
 >
 <
  CFCOL
  HEADER="Extension"
  TEXT="Ext. #PhoneExtension#"
 >
</CFTABLE>

</BODY>

</HTML>
```

To create the table, you use the tag `<CFTABLE QUERY="Employees" COLHEADERS>`. The CFTABLE tag is a special type of CFOUTPUT and, therefore, requires that you specify a QUERY attribute, just like the one you would provide to CFOUTPUT. You use CFTABLE only to display query results, and the QUERY attribute specifies which result set to process.

You use the COLHEADERS attribute to instruct Cold Fusion to create optional column headers for each column in the table.

Next, Cold Fusion needs to know what columns you want to include in your table. You specify each column by using the CFCOL tag. You specify two columns here, one for the employee name and one for the phone extension.

The code for the phone extension column is

```
<CFCOL  HEADER="Extension" TEXT="Ext. #PhoneExtension#">
```

The HEADER attribute specifies the text to use in the column header. This column has a header with the text *Extension* in it. The TEXT attribute is required; every CFCOL tag must have one. It tells Cold Fusion what you want to display in this column. The TEXT attribute here contains the expression `"Ext. #PhoneExtension#"`. As Cold Fusion processes each row, it replaces the `#PhoneExtension#` field with the value of the PhoneExtension column retrieved.

The employee name column may look more complicated, but it really isn't at all. The source for the column is

```
<CFCOL HEADER="Employee" TEXT="<A
HREF=""empdtl1.cfm?EmployeeID=#EmployeeID#"">#LastName#, #FirstName#</A>">
```

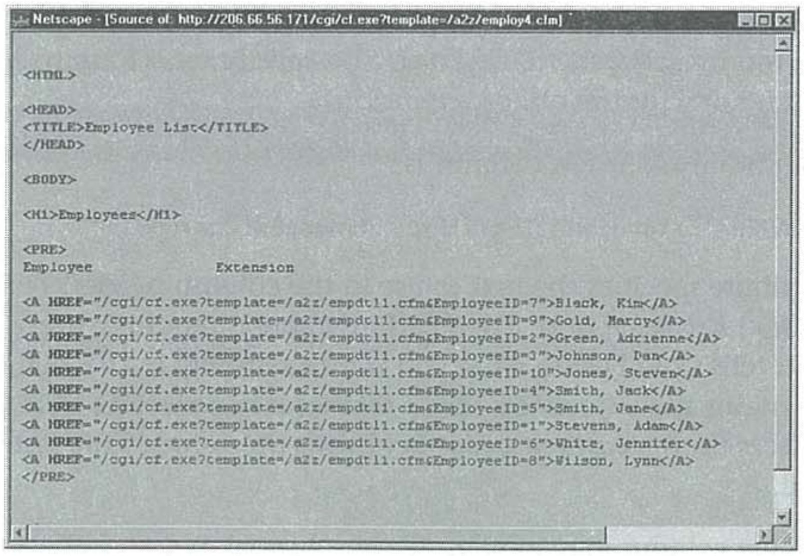
Again, you first specify the text for the optional header in the HEADER attribute. The TEXT attribute contains the text to display, and because the name has to be a hyperlink, you must specify the A HREF link tag, too.

In fact, the contents of the TEXT attribute are almost the same as the hyperlink tag you used in Listing 10.7 earlier—with one notable exception. Notice that the link tag has double quotation marks around the URL instead of the usual single set of quotation marks. You need the double quotation marks to tell Cold Fusion to treat this as a quote, not as the end of the TEXT attribute. If you were to enter a single quotation mark, Cold Fusion would think that the TEXT attribute ends right after the HREF=. And because it would not know what to do with the text after the quotation mark, Cold Fusion would report a syntax error. This process of using double quotes to indicate an actual quote character is called *escaping*, and the quote character is said to have been *escaped*.

So, now that you understand the code listing, go ahead and run the template. Go to URL **http://yourserver.com/ a2z/employ2.cfm**. As you can see in Figure 10.11, the employee names and phone extensions are now displayed in clearly labeled columns.

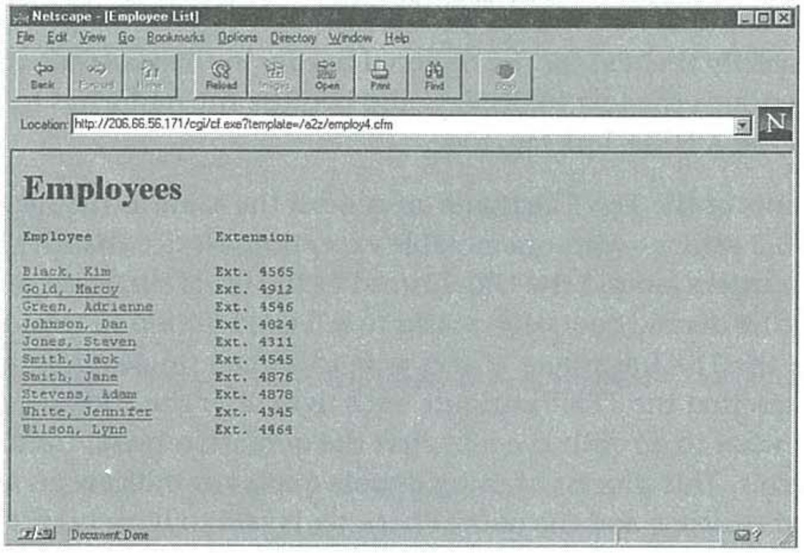
How is this table created without using the HTML `<TABLE>` tag? Look at the source code generated by Cold Fusion to find out. Select the View Source option in your browser (in Netscape, choose Document Source from the View menu; in Microsoft Internet Explorer, choose Source from the View menu).

As you can see in Figure 10.12, Cold Fusion uses the HTML `<PRE>` tag, which displays text exactly as it appears in the source code. Usually, Web browsers ignore white-space characters, such as spaces and line feeds. The `<PRE>` tag instructs the browser to maintain all spacing and line feeds, allowing Cold Fusion to lay out the data exactly as it wants the browser to display it.


Netscape - [Source of: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ4.cfm]

```
<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<PRE>
Employee              Extension

<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=7">Black, Kim</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=9">Gold, Marcy</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=2">Green, Adrienne</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=3">Johnson, Dan</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=10">Jones, Steven</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=4">Smith, Jack</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=5">Smith, Jane</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=1">Stevens, Adam</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=6">White, Jennifer</A>
<A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=8">Wilson, Lynn</A>
</PRE>
```

Figure 10.11 Cold Fusion can generate non-HTML tables using the CFTABLE tag.


Netscape - [Employee List]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ4.cfm
Document: Done

Employees

```
Employee              Extension

Black, Kim            Ext. 4565
Gold, Marcy           Ext. 4912
Green, Adrienne       Ext. 4546
Johnson, Dan          Ext. 4824
Jones, Steven         Ext. 4311
Smith, Jack           Ext. 4545
Smith, Jane           Ext. 4876
Stevens, Adam         Ext. 4878
White, Jennifer       Ext. 4345
Wilson, Lynn          Ext. 4464
```

Figure 10.12 To see how Cold Fusion interprets your template, view the generated markup language code with your browser's View Source option.

Tip

Viewing the source code generated by Cold Fusion is useful in debugging template problems. When you view the source, you are looking at the complete output as it was sent to your browser. If you ever need to ascertain why a Web page does not look like you intended, a good place to start is comparing your template with the source code it generated.

Creating HTML Tables with CFTABLE

Tables created with the HTML `<TABLE>` tag, of course, look much better. So Cold Fusion supports HTML tables, too. As you can see in Listing 10.11, to create HTML tables, you just need to specify the HTMLTABLE attribute in the CFTABLE tag.

Listing 10.11 EMPLOY5.CFM—Create HTML Tables with CFTABLE

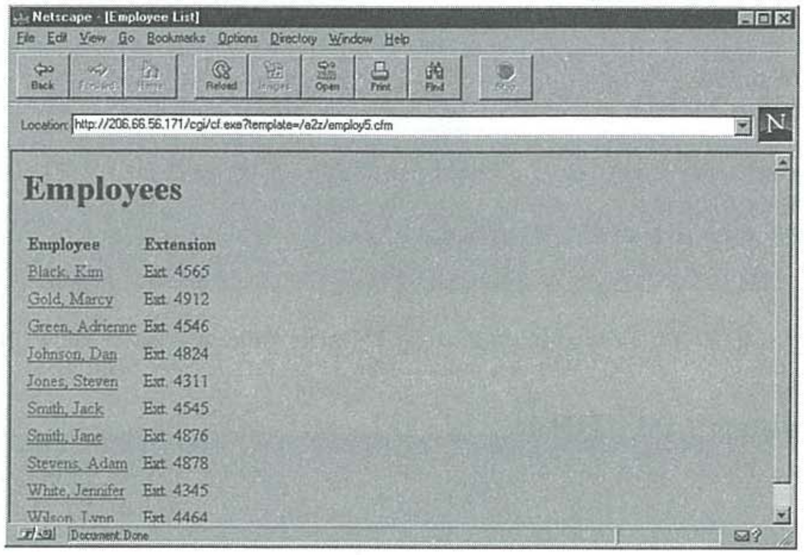
On the CD

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employees"
>
SELECT FirstName, LastName, PhoneExtension, EmployeeID
      FROM Employees
      ORDER BY LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<
 CFTABLE
 QUERY="Employees"
 COLHEADERS
 HTMLTABLE
>
 <
  CFCOL
  HEADER="Employee"
  TEXT="<A HREF=""empdtl1.cfm?EmployeeID=#EmployeeID#"">#LastName#,
#FirstName#</A>"
 >
 <
  CFCOL
  HEADER="Extension"
  TEXT="Ext. #PhoneExtension#"
 >
</CFTABLE>

</BODY>

</HTML>
```

Figure 10.13 shows the same employee list screen rendered in an HTML table. Note that when you're displaying data in an HTML table, standard fonts are used, not the fixed font used when the `<PRE>` tag is specified. Therefore, you can safely use any other HTML formatting options in the CFCOL TEXT attribute if required. If you want the name in bold, for example, you can specify

```
TEXT="<A HREF=""empdtl1.cfm?EmployeeID=#EmployeeID#""><B>#LastName#, #FirstName#</
B></A>"
```

And Cold Fusion still can display the table correctly. The `<B>` and `</B>` tags are HTML tags, not CFML tags, so Cold Fusion just passes them through to the Web server to be sent to your Web browser.

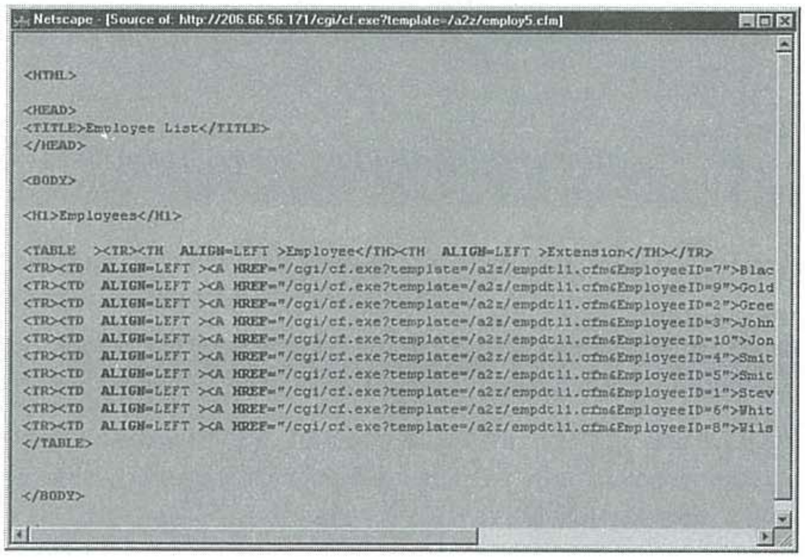

Netscape - [Employee List]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ5.cfm
Document: Done

Employees

Employee	Extension
Black, Kim	Ext. 4565
Gold, Marcy	Ext. 4912
Green, Adrienne	Ext. 4546
Johnson, Dan	Ext. 4824
Jones, Steven	Ext. 4311
Smith, Jack	Ext. 4545
Smith, Jane	Ext. 4876
Stevens, Adam	Ext. 4878
White, Jennifer	Ext. 4345
Wilson, Lynn	Ext. 4464

Figure 10.13 You can use the CFTABLE tag to create HTML tables.

To create this table, Cold Fusion generates HTML table code. This source code, as displayed by the browser's view source function, is shown in Figure 10.14.


Netscape - [Source of: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ5.cfm]

```
<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<TABLE  ><TR><TH  ALIGN=LEFT >Employee</TH><TH  ALIGN=LEFT >Extension</TH></TR>
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=7">Blac
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=9">Gold
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=2">Gree
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=3">John
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=10">Jon
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=4">Smit
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=5">Smit
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=1">Stev
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=6">Whit
<TR><TD  ALIGN=LEFT ><A HREF="/cgi/cf.exe?template=/a2z/empdtl1.cfm&EmployeeID=8">Wils
</TABLE>


</BODY>
```

Figure 10.14 Cold Fusion can generate all the required code to create HTML tables.

Creating HTML Tables Manually

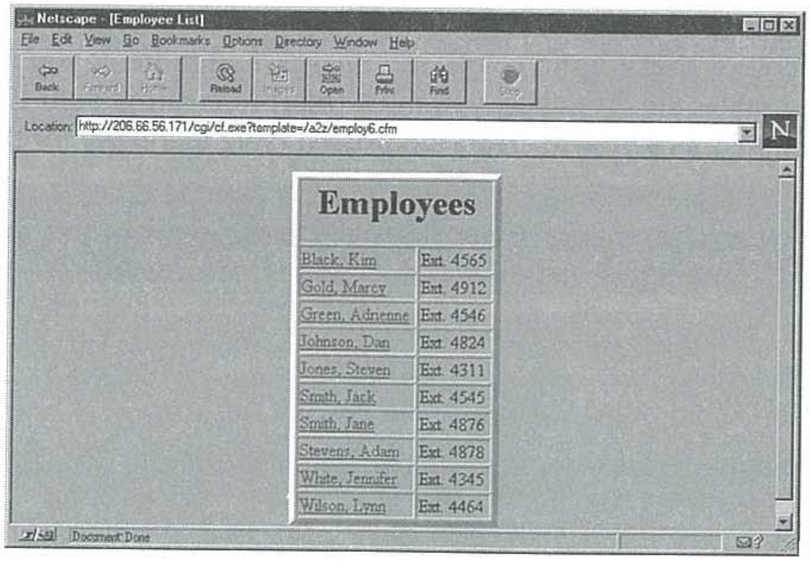
As good as the Cold Fusion `<CFTABLE>` tag is, it is very limited. HTML tables support many advanced features such as table headers, cells that span multiple rows or columns, borders and border colors, background colors and images, and more. If you really want to control how your tables are displayed, you must resort to creating your tables manually. Listing 10.12 demonstrates how to create a bordered table manually for the employee list.

Listing 10.12 EMPLOY6.CFM—Creating Tables Manually

On the CD

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employees"
>
SELECT FirstName, LastName, PhoneExtension, EmployeeID
      FROM Employees
      ORDER BY LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<CENTER>

<TABLE BORDER=5>

<TR>
 <TH COLSPAN=2>
  <H1>Employees</H1>
 </TH>
</TR>

<CFOUTPUT QUERY="Employees">
 <TR>
  <TD>
   <A HREF="empdtl1.cfm?EmployeeID=#EmployeeID#">#LastName#, #FirstName#</A>
  </TD>
  <TD>
   Ext. #PhoneExtension#
  </TD>
 </TR>
</CFOUTPUT>

</TABLE>

</CENTER>

</BODY>

</HTML>
```

Figure 10.15 shows the output for this listing.


Netscape - [Employee List]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ6.cfm
Document: Done

Employees	
Black, Kim	Ext. 4565
Gold, Marcy	Ext. 4912
Green, Adrienne	Ext. 4546
Johnson, Dan	Ext. 4824
Jones, Steven	Ext. 4311
Smith, Jack	Ext. 4545
Smith, Jane	Ext. 4876
Stevens, Adam	Ext. 4878
White, Jennifer	Ext. 4345
Wilson, Lynn	Ext. 4464

Figure 10.15 Creating tables manually gives you a greater degree of control over table appearance.

Now look at the code in Listing 10.12. First, you create the table with the `<TABLE>` tag and specify an optional border. HTML tables can have borders of varying thicknesses, and the BORDER attribute specifies the border to use. Then you create a table title and place it in a header cell (specified with the `<TH>` tag) that spans two columns.

Next comes the CFOUTPUT. As each query row is output, a new table row is created. For this reason, you include a complete table row (the `<TR>` tag) and cells (the `<TD>` tag) within the CFOUTPUT code block. And finally, you close the table with a `</TABLE>` tag.

As you can see, manually creating tables requires more effort and a better understanding of HTML tables, but the rewards are well worth your time.

Note

Using HTML tables is a useful way to format data, but a cost is associated with using tables. For a browser to display a table correctly, it cannot display any part of that table until it receives the entire table from the Web server. This happens because any row, even one near the end of the table, can have an effect on the width of columns and how the table is formatted. Therefore, if you display data in a table, the user doesn't see any data at all until all the data is present. If you use another type of display—for example, a list—the data is displayed as it is received. The reality is, the page may likely take as long to fully load with or without tables. The downside of using tables is that it takes longer for any data to appear. This, however, does not apply to tables created without the `<TABLE>` tag.

Grouping Query Results

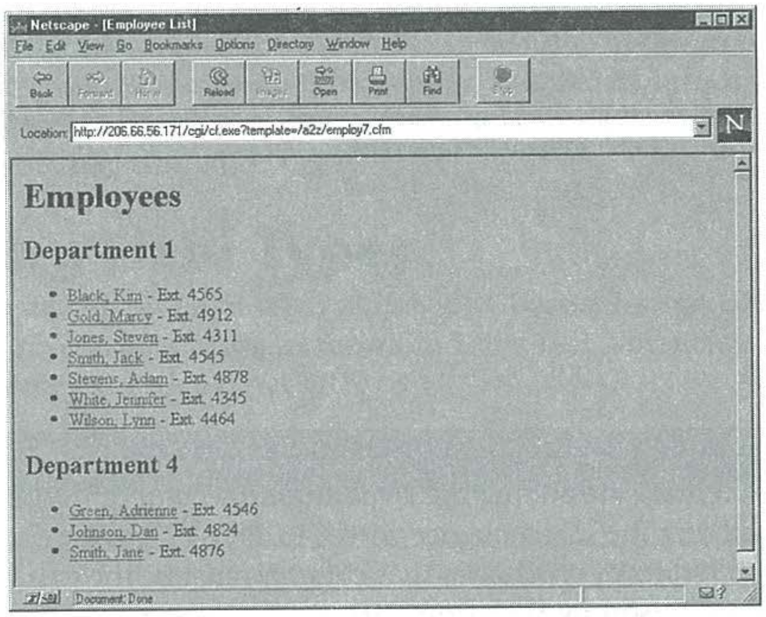
Before I introduce a new level of complexity, let me review how Cold Fusion processes queries for you. In Cold Fusion, you create data queries by using the `<CFQUERY>` tag.

CFQUERY performs an SQL operation and retrieves results, if any. Results are stored temporarily by Cold Fusion, and they remain around only for the duration of the processing of the template that contains the query.

To output query results, you use the `<CFOUTPUT>` tag. CFOUTPUT takes a query name as an attribute and then loops through all the rows that are retrieved by the query. The code block between the `<CFOUTPUT>` and the `</CFOUTPUT>` is repeated once for every row retrieved.

All the examples you created to this point displayed results in a single list or a single table. But what do you do if you want to process the results in subsets? For example, suppose you want to list the employees by department. You could just change the SQL statement in the CFQUERY to set the sort order to be department and then, perhaps, by name within each department.

This approach, however, would retrieve the data in the correct order, but how would you display it? If you use CFOUTPUT like you have until now, then every row created by the CFOUTPUT block has to be the same. If one has a department name, then all have to, because every row that is processed is processed with the same block of code. So how do you create the output shown in Figure 10.16?


Netscape - [Employee List]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://206.66.56.171/cgi/cf.exe?template=/a2z/employ7.cfm
Document: Done

Employees

Department 1

- Black, Kim - Ext. 4565
- Gold, Marcy - Ext. 4912
- Jones, Steven - Ext. 4311
- Smith, Jack - Ext. 4545
- Stevens, Adam - Ext. 4878
- White, Jennifer - Ext. 4345
- Wilson, Lynn - Ext. 4464

Department 4

- Green, Adrienne - Ext. 4546
- Johnson, Dan - Ext. 4824
- Smith, Jane - Ext. 4876

Figure 10.16 You can use the CFOUTPUT tag to group query results and display them accordingly.

The solution is to group the data results. By grouping, you can have more than one CFOUTPUT loop. To understand how grouping works, look at the template in Listing 10.13.

On the CD

Listing 10.13 EMPLOY7.CFM—Employee List Grouped by Department

```
<
 CFQUERY
 DATASOURCE="A2Z"
 NAME="Employees"
>
SELECT DepartmentID, FirstName, LastName, PhoneExtension, EmployeeID
      FROM Employees
      ORDER BY DepartmentID, LastName, FirstName
</CFQUERY>

<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H1>Employees</H1>

<CFOUTPUT QUERY="Employees" GROUP="DepartmentID">
 <H2>Department #DepartmentID#</H2>

 <UL>

  <CFOUTPUT>
   <LI><A HREF="empdtl1.cfm?EmployeeID=#EmployeeID#">#LastName#,
#FirstName#</A> - Ext. #PhoneExtension#
  </CFOUTPUT>

 </UL>

</CFOUTPUT>

</BODY>

</HTML>
```

The first changes you make are adding the DepartmentID column to the SQL SELECT statement and modifying the sort sequence with ORDER BY DepartmentID, LastName, FirstName. To group results by a column, that column must be the first in the sort sequence. As you want to sort by DepartmentID, that column is now the first in the ORDER BY list.

The big change, however, is the CFOUTPUT block. You now have two of them, one nested inside the other. The outer CFOUTPUT tag also has a new attribute: `GROUP="DepartmentID"`. A CFOUTPUT tag creates a loop that executes once for each row retrieved by a query. When you add the GROUP attribute, you instruct Cold Fusion to execute the CFOUPUT block only when the group value changes.

If you have seven employees with the exact same DepartmentID, the GROUP CFOUTPUT block is executed just once. In the list, you have 10 employees who work in two departments. The outer CFOUPUT block gets executed twice, once for each department. The first row processed has a DepartmentID of 2, so the CFOUTPUT block is executed. The next six rows processed also have a DepartmentID of 2, so the CFOUTPUT block is not executed for them. The eighth row has a different DepartmentID, with a value of 4, so the CFOUTPUT block is executed. The next two rows also have a DepartmentID of 4, so no CFOUTPUT block is executed for them. That's just the outer CFOUTPUT block. The inner block gets executed for every row, just like the CFOUTPUT blocks you used earlier.

Now look at the output code in Listing 10.13. The outer CFOUTPUT creates a header for each new group and then starts a new unordered list. The inner CFOUTPUT populates that list until the group is completed. Then the outer CFOUTPUT terminates the list, and the process loops to the next DepartmentID. The results are shown in Figure 10.16.

Note

Groups may be nested by creating additional CFOUTPUT blocks, one for each group. There is no limit to the number of groups that may be nested as long as these two conditions are met. First, every group must be part of the sort sequence used to retrieve the data. Second, the order that the columns appear in the ORDER BY clause must match the order of the groupings.

Now you can see why the column you want to group on must be the first in the ORDER BY list. For grouping to work, all rows with the same value in the grouping column must be processed as a group. If the group is broken up, as could happen if you do not sort by the grouping column, Cold Fusion executes the outer block at the wrong times, and the resulting groups are fragmented.

Specifying Field Types

You have now used two different types of fields: CGI variables and URL parameters. Cold Fusion supports several field types, as shown in Table 10.1, and fields that are database table columns retrieved with a CFQUERY.

Table 10.1 Cold Fusion Field Types

Field	Description
CGI	HTTP CGI variables (see The CD, "CGI Environment Variables")
CLIENT	Client variables (see Chapter 23, "Web Application Framework")
COOKIE	HTTP client-side cookies (see the CD for "Persistent Client Cookies")
FORM	HTML form fields (see Chapter 11, "Cold Fusion Forms")
URL	Parameters passed to a URL
VARIABLES	Cold Fusion variables (see Chapter 17, "Advanced Cold Fusion Templates")

In this chapter, you used two of these field types and CFQUERY results. By the time you finish reading this book, you'll be using them all regularly. And sooner or later, you're going to run into a name collision. For example, you may have a form field with the exact same name as a table column or a variable with the same name as a URL parameter. When this situation occurs, how does Cold Fusion know which one to use? Well, the answer is that Cold Fusion doesn't know. You must specify which to use. And the way you specify is by qualifying the field name with the field type.

Listing 10.14 is the same template you created in Listing 10.3, with one difference. The references to field name are fully qualified as URL.name. This way, even if you have any other field called name, Cold Fusion still knows which field you are referring to.

Listing 10.14 HELLO4.CFM—Avoiding Name Conflicts

```
<HTML>

<HEAD>
<TITLE>Hello!</TITLE>
</HEAD>

<BODY>

Hello,<BR>

<CFIF #ParameterExists(URL.name)# IS "Yes">
 <CFOUTPUT>
 The name you entered is <B>#URL.name#</B>
 </CFOUTPUT>
<CFELSE>
 You did not pass a parameter called NAME
</CFIF>


</BODY>

</HTML>
```

Use your browser to view this template. The resulting display should be exactly the same as shown in Figure 10.3.

From Here...

The information in this chapter is very important. I strongly urge you not to go any further until you understand all the examples presented here. Cold Fusion field processing, queries, and output form the basis for almost any application you will develop.

In this chapter, you created your first real Cold Fusion application. I introduced the CFQUERY and CFOUTPUT tags and explained examples of each. You learned how to use Cold Fusion fields and how to pass fields as parameters to a URL. You also learned how to create a drill-down application and how you can use HTML frames to make drill down even more usable. You also learned about the Cold Fusion CFTABLE tag and how to create dynamic tables with and without this tag. You learned how to group output results into organized and logical data sets. And finally, you learned how to specify field types to prevent field name collisions.

For more information about topics mentioned in this chapter, see the following chapters:

- Chapter 9, "SQL Data Manipulation," teaches you three other important SQL statements: the INSERT statement used to add new rows to a table, the UPDATE statement used to modify one or more rows in a table, and the DELETE statement used to delete one or more rows from a table.
- Chapter 11, "Cold Fusion Forms," builds on what you learned here by showing you how you can use forms to search for specific data.
- Chapter 13, "Web Application Wizards," teaches you how to jump-start the development process by using the Application Wizards to create a base from which you can build your application.
- Chapter 16, "Advanced SQL," teaches the use of advanced SQL concepts, including views and table joins.
- Chapter 17, "Advanced Cold Fusion Templates," teaches the use of advanced Cold Fusion template techniques to help you get the most out of your application development effort.