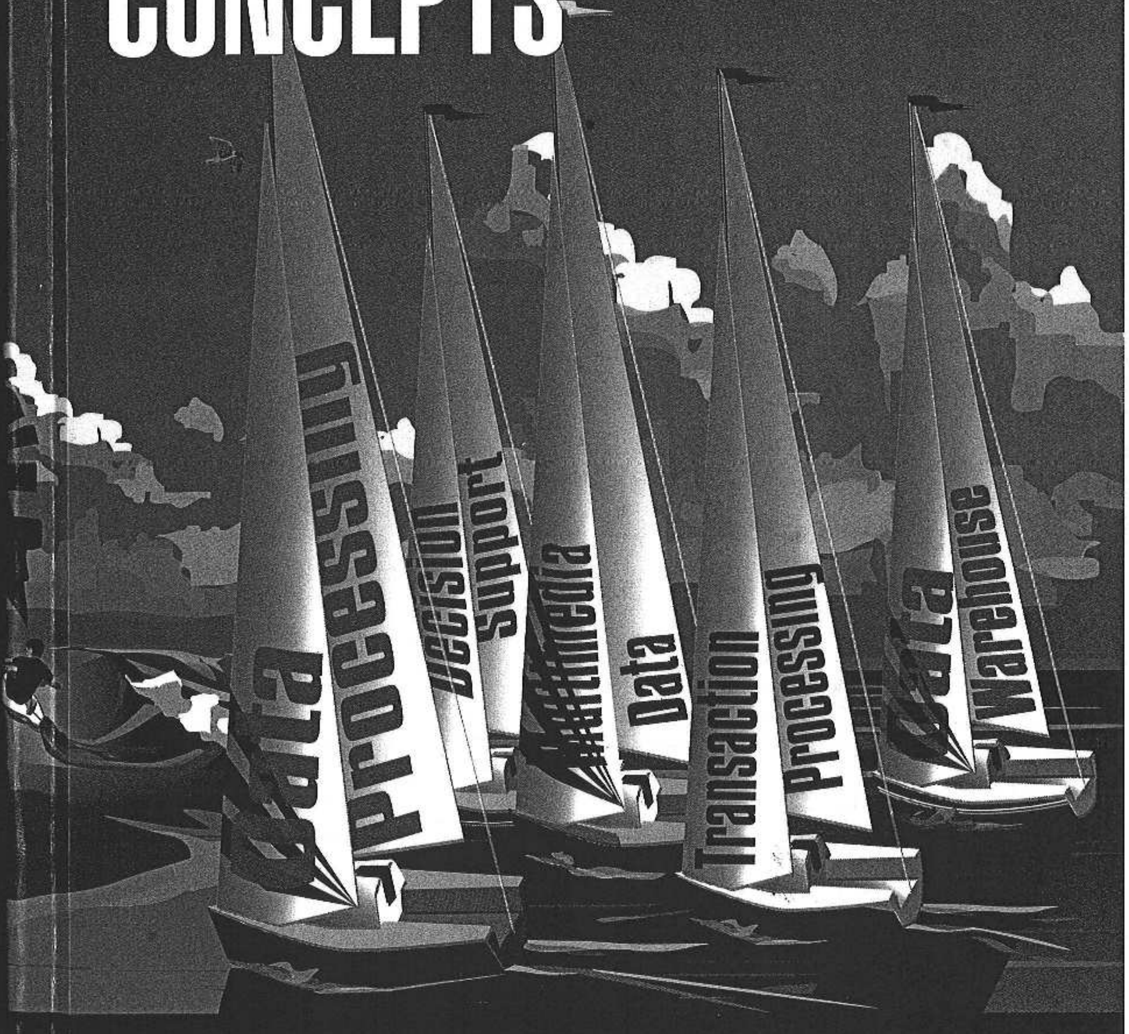
THIRD EDITION

# DATABASE SYSTEM CONCEPTS



ABRAHAM SILBERSCHATZ HENRY F. KORTH
S. SUDARSHAN

*McGraw-Hill*
*A Division of The* ***McGraw-Hill*** *Companies*

**DATABASE SYSTEM CONCEPTS**

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 0 DOC DOC 9 0 9 8 7 6

ISBN 0-07-044756-X

This book was set in Times Roman by ETP Harrison.
The editor was Eric M. Munson;
the production supervisor was Leroy A. Young.
The cover was designed by Christopher Brady.
Project supervision was done by ETP Harrison (Portland, Oregon).
R. R. Donnelley & Sons Company was printer and binder.

**Library of Congress Cataloging-in-Publication Data**

Silberschatz, Abraham.
Database system concepts / Abraham Silberschatz, Henry F. Korth, S. Sudarshan. — 3rd ed.
p. cm. — (McGraw-Hill series in computer science)
Korth's name appears first on earlier editions.
Includes bibliographical references and index.
ISBN 0-07-044756-X
1. Database management. I. Korth, Henry F. II. Sudarshan, S. III. Title. IV. Series: McGraw-Hill computer science series.
QA76.9.D3S5737 1996
005.74—dc21 96-44015

**http://www.mhcollege.com**

# CHAPTER 1

# INTRODUCTION

A *database-management system* (DBMS) consists of a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the *database*, contains information about one particular enterprise. The primary goal of a DBMS is to provide an environment that is both *convenient* and *efficient* to use in retrieving and storing database information.

Database systems are designed to manage large bodies of information. The management of data involves both the definition of structures for the storage of information and the provision of mechanisms for the manipulation of information. In addition, the database system must provide for the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

The importance of information in most organizations—which determines the value of the database—has led to the development of a large body of concepts and techniques for the efficient management of data. In this chapter, we present a brief introduction to the principles of database systems.

## 1.1 Purpose of Database Systems

Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. One way to keep the information on a computer is to store it in permanent system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including

- A program to debit or credit an account
- A program to add a new account

- A program to find the balance of an account
- A program to generate monthly statements

These application programs have been written by system programmers in response to the needs of the bank organization.

New application programs are added to the system as the need arises. For example, suppose that new government regulations allow the savings bank to offer checking accounts. As a result, new permanent files are created that contain information about all the checking accounts maintained in the bank, and new application programs may need to be written to deal with situations that do not arise in savings accounts, such as handling overdrafts. Thus, as time goes by, more files and more application programs are added to the system.

The typical *file-processing system* just described is supported by a conventional operating system. Permanent records are stored in various files, and different application programs are written to extract records from, and to add records to, the appropriate files. Before the advent of DBMSs, organizations typically stored information using such systems.

Keeping organizational information in a file-processing system has a number of major disadvantages

- **Data redundancy and inconsistency**. Since the files and application programs are created by different programmers over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to *data inconsistency*; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.
- **Difficulty in accessing data**. Suppose that one of the bank officers needs to find out the names of all customers who live within the city's 78733 zip code. The officer asks the data-processing department to generate such a list. Because this request was not anticipated when the original system was designed, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* customers. The bank officer has now two choices: Either obtain the list of all customers and have the needed information extracted manually, or ask the data-processing department to have a system programmer write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same officer needs to trim that list to include only those customers who have an account balance of $10,000 or more. As expected, a program to generate such a list does not exist. Again, the officer has the preceding two options, neither of which is satisfactory.

The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems must be developed for general use.

- **Data isolation**. Because data are scattered in various files, and files may be in different formats, it is difficult to write new application programs to retrieve the appropriate data.
- **Integrity problems**. The data values stored in the database must satisfy certain types of *consistency constraints*. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.
- **Atomicity problems**. A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial to ensure that, once a failure has occurred and has been detected, the data are restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account *A* to *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure this property in a conventional file-processing system.
- **Concurrent-access anomalies**. So that the overall performance of the system is improved and a faster response time is possible, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350. To guard against this possibility, the system must maintain some form of supervision. Because data may be accessed by many different application programs that have not been coordinated previously, however, supervision is difficult to provide.
- **Security problems**. Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer

# Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

## Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

## Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

## Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

## API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

**LAW FIRMS**
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

**FINANCIAL INSTITUTIONS**
Litigation and bankruptcy checks for companies and debtors.

**E-DISCOVERY AND LEGAL VENDORS**
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.