Part 1

Introduction to Database Environments

Part 1 provides a background for the subsequent detailed study of database design, database application development, and database administration. The chapters in Part 1 present the principles of database management and the nature of the database development process. Chapter 1 covers the basic concepts of database management including database characteristics, features and architectures of database management systems, the market for database management systems, and organizational impacts of database technology. Chapter 2 introduces the context, objectives, phases, and tools of the database development process.

Chapter 1

Introduction to Database Management

Learning Objectives

This chapter provides an introduction to database technology and the impact of this technology on organizations. After this chapter the student should have acquired the following knowledge and skills:

- Describe the characteristics of business databases and the features of database management systems.
- Understand the importance of nonprocedural access for software productivity.
- Appreciate the advances in database technology and the contributions of database technology to modern society.
- Understand the impact of database management system architectures on distributed processing and software maintenance.
- Perceive career opportunities related to database application development and database administration.

Overview

You may not be aware of it, but your life is dramatically affected by database technology. Computerized databases are vital to the functioning of modern organizations. You come into contact with databases on a daily basis through activities such as shopping at a supermarket, withdrawing cash using an automated teller machine, ordering a book online, and registering for classes. The convenience of your daily life is partly due to proliferation of computerized databases and supporting database technology.

Database technology is not only improving the daily operations of organizations but also the quality of decisions that affect our lives. Databases contain a flood of data about many aspects of our lives: consumer preferences, telecommunications usage, credit history, television viewing habits, and so on. Database technology helps to summarize this mass of data into useful information for decision making. Management uses information gleaned from databases to make long-range decisions such as investing in plants and equipment, locating stores, adding new items to inventory, and entering new businesses.

This first chapter provides a starting point for your exploration of database technology. It surveys database characteristics, database management system features, system

architectures, and human roles in managing and using databases. The other chapter in Part 1 (Chapter 2) provides a conceptual overview of the database development process. This chapter provides a broad picture of database technology and shares the excitement about the journey ahead.

1.1 Database Characteristics

database
a collection of persistent data that can be shared and interrelated.

Every day, businesses collect mountains of facts about persons, things, and events such as credit card numbers, bank balances, and purchase amounts. Databases contain these sorts of simple facts as well as nonconventional facts such as photographs, fingerprints, product videos, and book abstracts. With the proliferation of the Internet and the means to capture data in computerized form, a vast amount of data is available at the click of a mouse button. Organizing these data for ease of retrieval and maintenance is paramount. Thus, managing databases has become a vital task in most organizations.

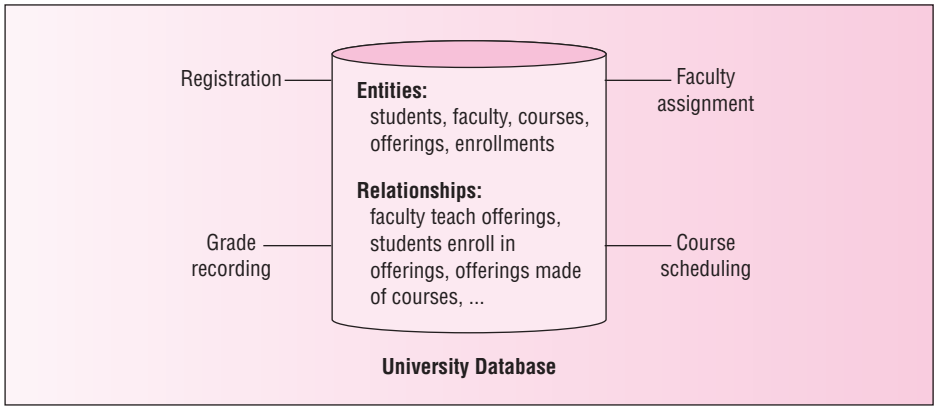
Before learning about managing databases, you must first understand some important properties of databases, as discussed in the following list:

- Persistent means that data reside on stable storage such as a magnetic disk. For example, organizations need to retain data about customers, suppliers, and inventory on stable storage because these data are repetitively used. A variable in a computer program is not persistent because it resides in main memory and disappears after the program terminates. Persistency does not mean that data lasts forever. When data are no longer relevant (such as a supplier going out of business), they are removed or archived.

  Persistency depends on relevance of intended usage. For example, the mileage you drive for work is important to maintain if you are self-employed. Likewise, the amount of your medical expenses is important if you can itemize your deductions or you have a medical savings account. Because storing and maintaining data is costly, only data likely to be relevant to decisions should be stored.
- Shared means that a database can have multiple uses and users. A database provides a common memory for multiple functions in an organization. For example, a personnel database can support payroll calculations, performance evaluations, government reporting requirements, and so on. Many users can access a database at the same time. For example, many customers can simultaneously make airline reservations. Unless two users are trying to change the same part of the database at the same time, they can proceed without waiting on each other.
- Interrelated means that data stored as separate units can be connected to provide a whole picture. For example, a customer database relates customer data (name, address, . . .) to order data (order number, order date, . . .) to facilitate order processing. Databases contain both entities and relationships among entities. An entity is a cluster of data usually about a single subject that can be accessed together. An entity can denote a person, place, thing, or event. For example, a personnel database contains entities such as employees, departments, and skills as well as relationships showing employee assignments to departments, skills possessed by employees, and salary history of employees. A typical business database may have hundreds of entities and relationships.

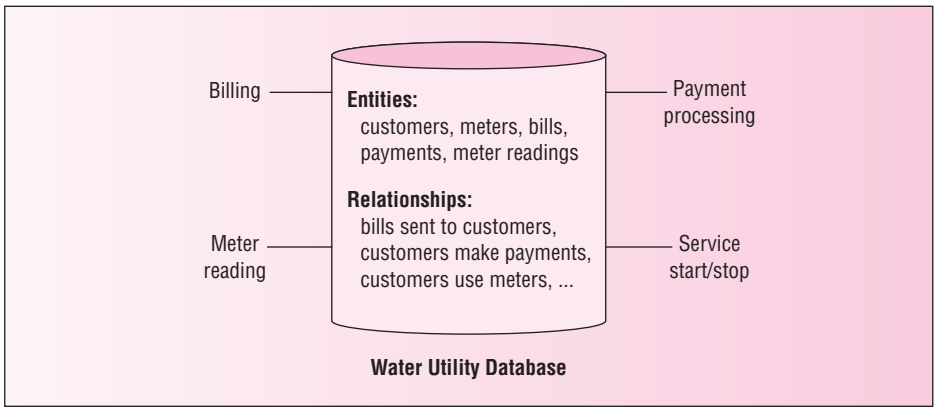
To depict these characteristics, let us consider a number of databases. We begin with a simple university database (Figure 1.1) since you have some familiarity with the workings of a university. A simplified university database contains data about students, faculty, courses, course offerings, and enrollments. The database supports procedures such as registering for classes, assigning faculty to course offerings, recording grades, and scheduling

FIGURE 1.1
Depiction of a Simplified University Database



Note: Words surrounding the database denote procedures that use the database.

FIGURE 1.2
Depiction of a Simplified Water Utility Database



course offerings. Relationships in the university database support answers to questions such as

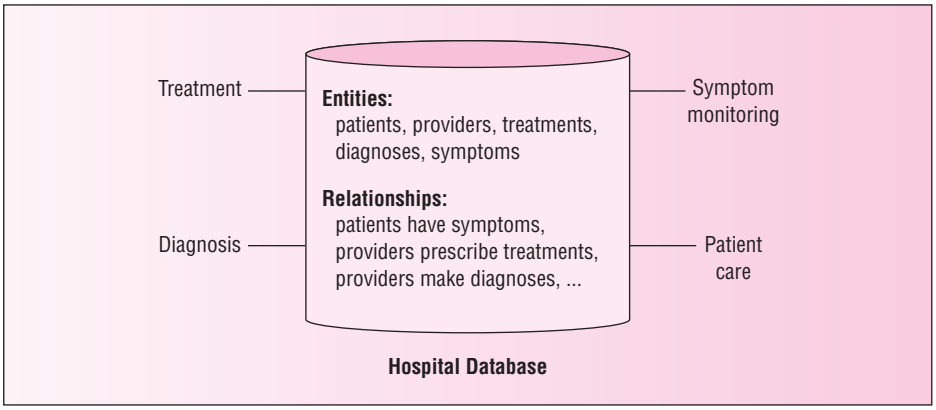
- What offerings are available for a course in a given academic period?
- Who is the instructor for an offering of a course?
- What students are enrolled in an offering of a course?

Next, let us consider a water utility database as depicted in Figure 1.2. The primary function of a water utility database is billing customers for water usage. Periodically, a customer's water consumption is measured from a meter and a bill is prepared. Many aspects can influence the preparation of a bill such as a customer's payment history, meter characteristics, type of customer (low income, renter, homeowner, small business, large business, etc.), and billing cycle. Relationships in the water utility database support answers to questions such as

- What is the date of the last bill sent to a customer?
- How much water usage was recorded when a customer's meter was last read?
- When did a customer make his/her last payment?

Finally, let us consider a hospital database as depicted in Figure 1.3. The hospital database supports treatment of patients by physicians. Physicians make diagnoses and prescribe treatments based on symptoms. Many different health providers read and contribute to a patient's medical record. Nurses are responsible for monitoring symptoms and providing

FIGURE 1.3
Depiction of a Simplified Hospital Database



medication. Food staff prepare meals according to a dietary plan. Physicians prescribe new treatments based on the results of previous treatments and patient symptoms. Relationships in the database support answers to questions such as

- What are the most recent symptoms of a patient?
- Who prescribed a given treatment of a patient?
- What diagnosis did a doctor make for a patient?

These simplified databases lack many kinds of data found in real databases. For example, the simplified university database does not contain data about course prerequisites and classroom capacities and locations. Real versions of these databases would have many more entities, relationships, and additional uses. Nevertheless, these simple databases have the essential characteristics of business databases: persistent data, multiple users and uses, and multiple entities connected by relationships.

1.2 Features of Database Management Systems

database management system (DBMS)
a collection of components that support data acquisition, dissemination, maintenance, retrieval, and formatting.

A database management system (DBMS) is a collection of components that supports the creation, use, and maintenance of databases. Initially, DBMSs provided efficient storage and retrieval of data. Due to marketplace demands and product innovation, DBMSs have evolved to provide a broad range of features for data acquisition, storage, dissemination, maintenance, retrieval, and formatting. The evolution of these features has made DBMSs rather complex. It can take years of study and use to master a particular DBMS. Because DBMSs continue to evolve, you must continually update your knowledge.

To provide insight about features that you will encounter in commercial DBMSs, Table 1.1 summarizes a common set of features. The remainder of this section presents examples of these features. Some examples are drawn from Microsoft Access, a popular desktop DBMS. Later chapters expand upon the introduction provided here.

1.2.1 Database Definition

table
a named, two-dimensional arrangement of data. A table consists of a heading part and a body part.

To define a database, the entities and relationships must be specified. In most commercial DBMSs, tables store collections of entities. A table (Figure 1.4) has a heading row (first row) showing the column names and a body (other rows) showing the contents of the table. Relationships indicate connections among tables. For example, the relationship connecting the student table to the enrollment table shows the course offerings taken by each student.

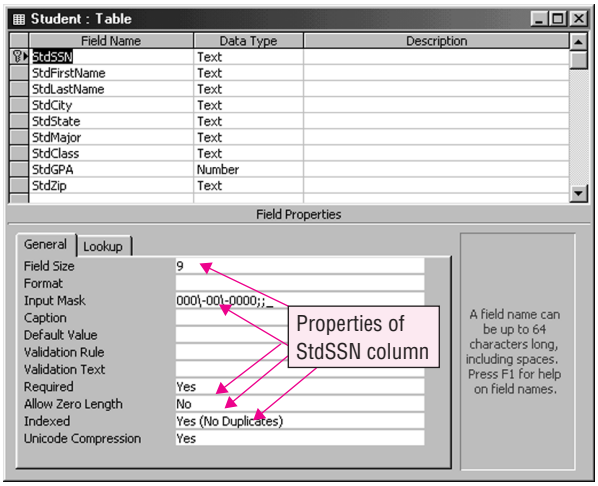
TABLE 1.1
Summary of Common Features of DBMSs

Feature	Description
Database definition	Language and graphical tools to define entities, relationships, integrity constraints, and authorization rights
Nonprocedural access	Language and graphical tools to access data without complicated coding
Application development	Graphical tools to develop menus, data entry forms, and reports; data requirements for forms and reports are specified using nonprocedural access
Procedural language interface	Language that combines nonprocedural access with full capabilities of a programming language
Transaction processing	Control mechanisms to prevent interference from simultaneous users and recover lost data after a failure
Database tuning	Tools to monitor and improve database performance

FIGURE 1.4 **Display of Student Table in Microsoft Access**

StdFirstName	StdLastName	StdCity	StdState	StdZip	StdMajor	StdClass	StdGPA
HOMER	WELLS	SEATTLE	WA	98121-1111	IS	FR	3.00
BOB	NORBERT	BOTHELL	WA	98011-2121	FIN	JR	2.70
CANDY	KENDALL	TACOMA	WA	99042-3321	ACCT	JR	3.50
WALLY	KENDALL	SEATTLE	WA	98123-1141	IS	SR	2.80
JOE	ESTRADA	SEATTLE	WA	98121-2333	FIN	SR	3.20
MARIAH	DODGE	SEATTLE	WA	98114-0021	IS	JR	3.60
TESS	DODGE	REDMOND	WA	98116-2344	ACCT	SO	3.30

FIGURE 1.5
Table Definition Window in Microsoft Access

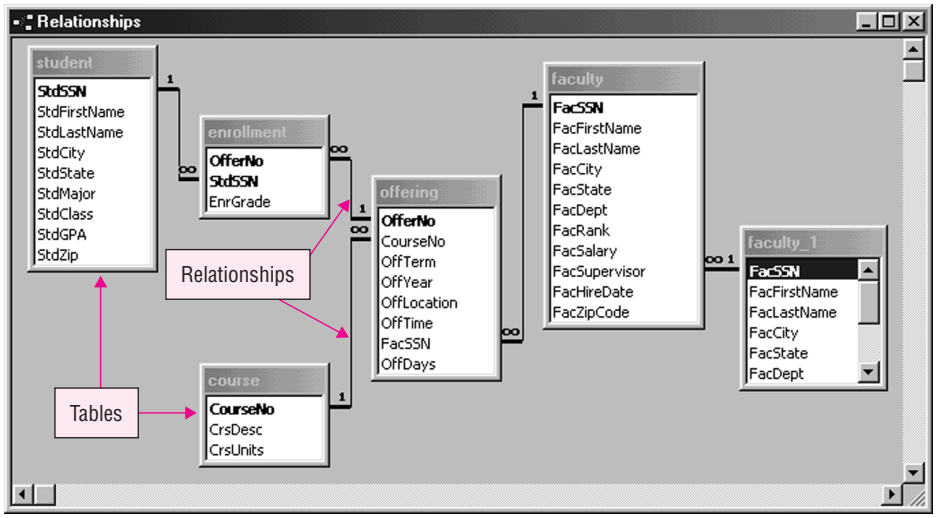


SQL
an industry standard database language that includes statements for database definition, database manipulation, and database control.

Most DBMSs provide several tools to define databases. The Structured Query Language (SQL) is an industry standard language supported by most DBMSs. SQL can be used to define tables, relationships among tables, integrity constraints (rules that define allowable data), and authorization rights (rules that restrict access to data). Chapter 3 describes SQL statements to define tables and relationships.

In addition to SQL, many DBMSs provide graphical, window-oriented tools. Figures 1.5 and 1.6 depict graphical tools for defining tables and relationships. Using the Table Definition window in Figure 1.5, the user can define properties of columns such as the data

FIGURE 1.6
Relationship Definition Window in Microsoft Access



type and field size. Using the Relationship Definition window in Figure 1.6, relationships among tables can be defined. After defining the structure, a database can be populated. The data in Figure 1.4 should be added after the Table Definition window and Relationship Definition window are complete.

1.2.2 Nonprocedural Access

nonprocedural database language
a language such as SQL that allows you to specify the parts of a database to access rather than to code a complex procedure. Nonprocedural languages do not include looping statements.

The most important feature of a DBMS is the ability to answer queries. A query is a request for data to answer a question. For example, the user may want to know customers having large balances or products with strong sales in a particular region. Nonprocedural access allows users with limited computing skills to submit queries. The user specifies the parts of a database to retrieve, not implementation details of how retrieval occurs. Implementation details involve coding complex procedures with loops. Nonprocedural languages do not have looping statements (for, while, and so on) because only the parts of a database to retrieve are specified.

Nonprocedural access can reduce the number of lines of code by a factor of 100 as compared to procedural access. Because a large part of business software involves data access, nonprocedural access can provide a dramatic improvement in software productivity.

To appreciate the significance of nonprocedural access, consider an analogy to planning a vacation. You specify your destination, travel budget, length of stay, and departure date. These facts indicate the "what" of your trip. To specify the "how" of your trip, you need to indicate many more details such as the best route to your destination, the most desirable hotel, ground transportation, and so on. Your planning process is much easier if you have a professional to help with these additional details. Like a planning professional, a DBMS performs the detailed planning process to answer queries expressed in a nonprocedural language.

Most DBMSs provide more than one tool for nonprocedural access. The SELECT statement of SQL, described in Chapter 4, provides a nonprocedural way to access a database. Most DBMSs also provide graphical tools to access databases. Figure 1.7 depicts a graphical tool available in Microsoft Access. To pose a query to the database, a user only has to indicate the required tables, relationships, and columns. Access is responsible for generating the plan to retrieve the requested data. Figure 1.8 shows the result of executing the query in Figure 1.7.

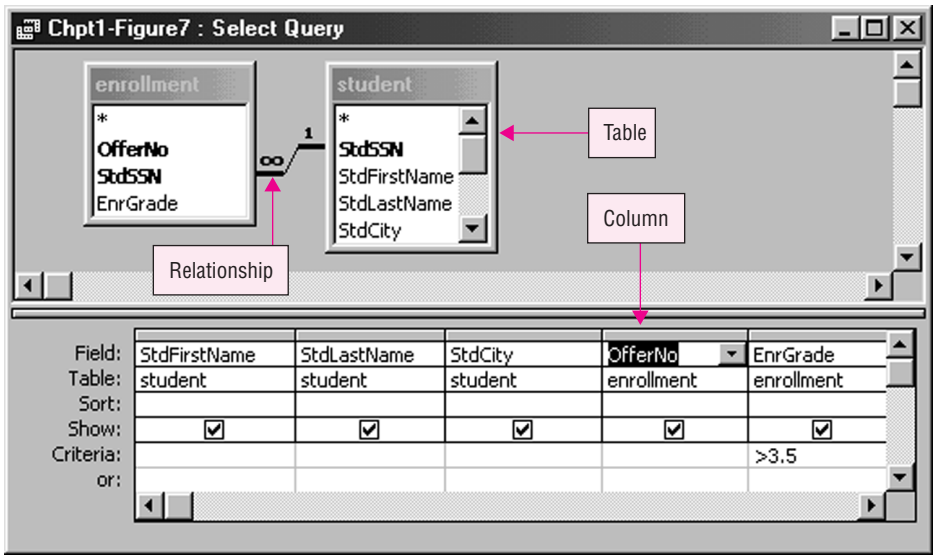


FIGURE 1.7
Query Design Window in Microsoft Access

StdFirstName	StdLastName	StdCity	OfferNo	EnrGrade
MARIAH	DODGE	SEATTLE	1234	3.8
BOB	NORBERT	BOTHELL	5679	3.7
ROBERTO	MORALES	SEATTLE	5679	3.8
MARIAH	DODGE	SEATTLE	6666	3.6
LUKE	BRAZZI	SEATTLE	7777	3.7
WILLIAM	PILGRIM	BOTHELL	9876	4

FIGURE 1.8
Result of Executing Query in Figure 1.7

1.2.3 Application Development and Procedural Language Interface

Most DBMSs go well beyond simply accessing data. Graphical tools are provided for building complete applications using forms and reports. Data entry forms provide a convenient tool to enter and edit data, while reports enhance the appearance of data that is displayed or printed. The form in Figure 1.9 can be used to add new course assignments for a professor and to change existing assignments. The report in Figure 1.10 uses indentation to show courses taught by faculty in various departments. The indentation style can be easier to view than the tabular style shown in Figure 1.8. Many forms and reports can be developed with a graphical tool without detailed coding. For example, Figures 1.9 and 1.10 were developed without coding. Chapter 10 describes concepts underlying form and report development.

Nonprocedural access makes form and report creation possible without extensive coding. As part of creating a form or report, the user indicates the data requirements using a nonprocedural language (SQL) or graphical tool. To complete a form or report definition, the user indicates formatting of data, user interaction, and other details.

procedural language interface
a method to combine a nonprocedural language such as SQL with a programming language such as COBOL or Visual Basic.

In addition to application development tools, a procedural language interface adds the full capabilities of a computer programming language. Nonprocedural access and application development tools, though convenient and powerful, are sometimes not efficient enough or do not provide the level of control necessary for application development. When these tools are not adequate, DBMSs provide the full capabilities of a programming language. For example, Visual Basic for Applications (VBA) is a programming language

FIGURE 1.9
Microsoft Access Form for Assigning Courses to Faculty

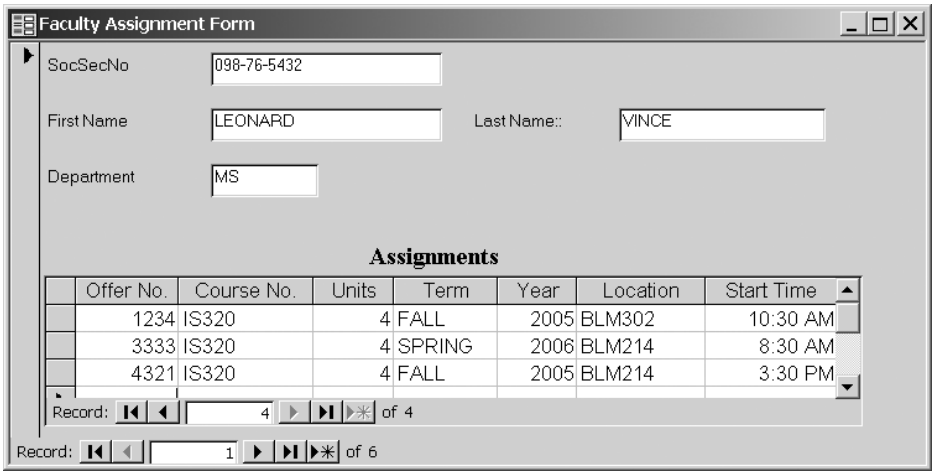
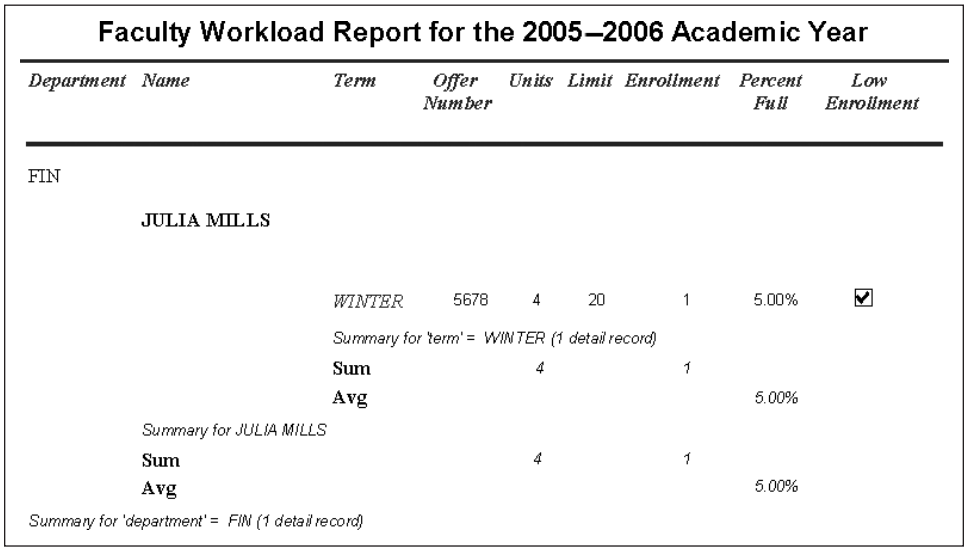


FIGURE 1.10
Microsoft Access Report of Faculty Workload



that is integrated with Microsoft Access. VBA allows full customization of database access, form processing, and report generation. Most commercial DBMSs have a procedural language interface comparable to VBA. For example, Oracle has the language PL/SQL and Microsoft SQL Server has the language Transact-SQL. Chapter 11 describes procedural language interfaces and the PL/SQL language.

1.2.4 Features to Support Database Operations

transaction processing
reliable and efficient processing of large volumes of repetitive work. DBMSs ensure that simultaneous users do not interfere with each other and that failures do not cause lost work.

Transaction processing enables a DBMS to process large volumes of repetitive work. A transaction is a unit of work that should be processed reliably without interference from other users and without loss of data due to failures. Examples of transactions are withdrawing cash at an ATM, making an airline reservation, and registering for a course. A DBMS ensures that transactions are free of interference from other users, parts of a transaction are not lost due to a failure, and transactions do not make the database inconsistent. Transaction processing is largely a "behind the scenes" affair. The user does not know the details about transaction processing other than the assurances about reliability.

Database tuning includes a number of monitors and utility programs to improve performance. Some DBMSs can monitor how a database is used, the distribution of various parts

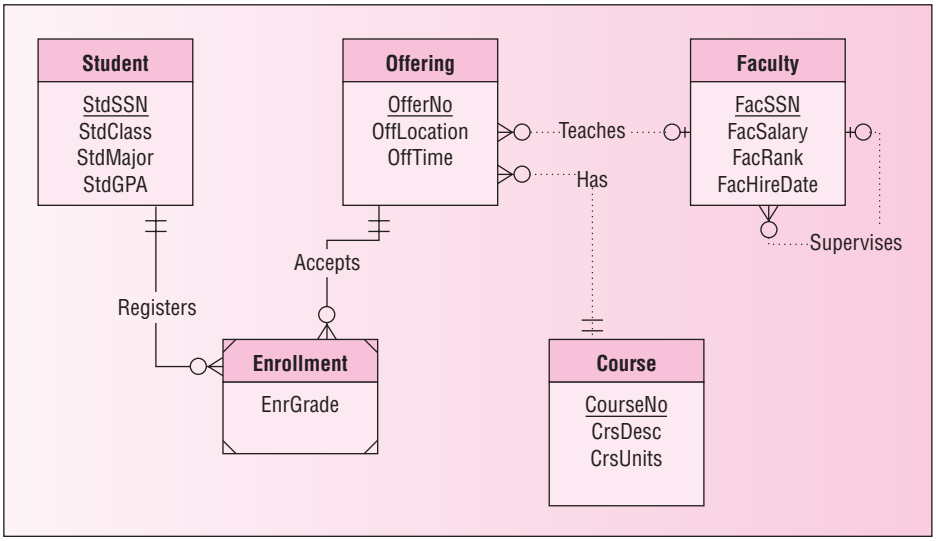


FIGURE 1.11
Entity Relationship Diagram (ERD) for the University Database

of a database, and the growth of the database. Utility programs can be provided to reorganize a database, select physical structures for better performance, and repair damaged parts of a database.

Transaction processing and database tuning are most prominent on DBMSs that support large databases with many simultaneous users. These DBMSs are known as enterprise DBMSs because the databases they support databases are often critical to the functioning of an organization. Enterprise DBMSs usually run on powerful servers and have a high cost. In contrast, desktop DBMSs running on personal computers and small servers support limited transaction processing features but have a much lower cost. Desktop DBMSs support databases used by work teams and small businesses. Embedded DBMSs are an emerging category of database software. As its name implies, an embedded DBMS resides in a larger system, either an application or a device such as a personal digital assistant (PDA) or a smart card. Embedded DBMSs provide limited transaction processing features but have low memory, processing, and storage requirements.

1.2.5 Third-Party Features

In addition to features provided directly by vendors of DBMSs, third-party software is also available for many DBMSs. In most cases, third-party software extends the features available with the database software. For example, many third-party vendors provide advanced database design tools that extend the database definition and tuning capabilities provided by DBMSs. Figure 1.11 shows a database diagram (an entity relationship diagram) created with Visio Professional, a tool for database design. The ERD in Figure 1.11 can be converted into the tables supported by most commercial DBMSs. In some cases, third-party software competes directly with the database product. For example, third-party vendors provide application development tools that can be used in place of the ones provided with the database product.

1.3 Development of Database Technology and Market Structure

The previous section provided a quick tour of the features found in typical DBMSs. The features in today's products are a significant improvement over just a few years ago. Database management, like many other areas of computing, has undergone tremendous technological growth. To provide you a context to appreciate today's DBMSs, this section reviews

TABLE 1.2
Brief Evolution of Database Technology

Era	Generation	Orientation	Major Features
1960s	1st generation	File	File structures and proprietary program interfaces
1970s	2nd generation	Network navigation	Networks and hierarchies of related records, standard program interfaces
1980s	3rd generation	Relational	Nonprocedural languages, optimization, transaction processing
1990s to 2000s	4th generation	Object	Multimedia, active, distributed processing, more powerful operators, data warehouse processing, XML enabled

past changes in technology and suggests future trends. After this review, the current market for database software is presented.

1.3.1 Evolution of Database Technology

Table 1.2 depicts a brief history of database technology through four generations[1] of systems. The first generation supported sequential and random searching, but the user was required to write a computer program to obtain access. For example, a program could be written to retrieve all customer records or to just find the customer record with a specified customer number. Because first-generation systems did not offer much support for relating data, they are usually regarded as file processing systems rather than DBMSs. File processing systems can manage only one entity rather than many entities and relationships managed by a DBMS.

The second-generation products were the first true DBMSs as they could manage multiple entity types and relationships. However, to obtain access to data, a computer program still had to be written. Second-generation systems are referred to as "navigational" because the programmer had to write code to navigate among a network of linked records. Some of the second-generation products adhered to a standard database definition and manipulation language developed by the Committee on Data Systems Languages (CODASYL), a standards organization. The CODASYL standard had only limited market acceptance partly because IBM, the dominant computer company during this time, ignored the standard. IBM supported a different approach known as the hierarchical data model.

Rather than focusing on the second-generation standard, research labs at IBM and academic institutions developed the foundations for a new generation of DBMSs. The most important development involved nonprocedural languages for database access. Third-generation systems are known as relational DBMSs because of the foundation based on mathematical relations and associated operators. Optimization technology was developed so that access using nonprocedural languages would be efficient. Because nonprocedural access provided such an improvement over navigational access, third-generation systems supplanted the second generation. Since the technology was so different, most of the new systems were founded by start-up companies rather than by vendors of previous generation products. IBM was the major exception. It was IBM's weight that led to the adoption of SQL as a widely accepted standard.

[1] The generations of DBMSs should not be confused with the generations of programming languages. In particular, fourth-generation language refers to programming language features, not DBMS features.

Fourth-generation DBMSs are extending the boundaries of database technology to unconventional data, the Internet, and data warehouse processing. Fourth-generation systems can store and manipulate unconventional data types such as images, videos, maps, sounds, and animations. Because these systems view any kind of data as an object to manage, fourth-generation systems are sometimes called "object-oriented" or "object-relational." Chapter 18 presents details about object features in DBMSs. In addition to the emphasis on objects, the Internet is pushing DBMSs to develop new forms of distributed processing. Most DBMSs now feature convenient ways to publish static and dynamic data on the Internet using the eXtensible Markup Language (XML) as a publishing standard. Chapter 17 presents details about client–server processing features in DBMSs to support Web access to databases.

A recent development in fourth-generation DBMSs is support for data warehouse processing. A data warehouse is a database that supports mid-range and long-range decision making in organizations. The retrieval of summarized data dominate data warehouse processing, whereas a mixture of updating and retrieving data occur for databases that support the daily operations of an organization. Chapter 16 presents details about DBMS features to support data warehouse processing.

The market for fourth-generation systems is a battle between vendors of third-generation systems who are upgrading their products against a new group of systems developed as open-source software. So far, the existing companies seem to have the upper hand.

1.3.2 Current Market for Database Software

According to the International Data Corporation (IDC), sales (license and maintenance) of enterprise database software reached $13.6 billion in 2003, a 7.6 percent increase since 2002. Enterprise DBMSs use mainframe servers running IBM's MVS operating system and mid-range servers running Unix (Linux, Solaris, AIX, and other variations) and Microsoft Windows Server operating systems. Sales of enterprise database software have followed economic conditions with large increases during the Internet boom years followed by slow growth during the dot-com and telecom slowdowns. For future sales, IDC projects sales of enterprise DBMSs to reach $20 billion by 2008.

According to IDC, three products dominate the market for enterprise database software as shown in Table 1.3. The IDC rankings include both license and maintenance revenues. When considering only license costs, the Gartner Group ranks IBM with the largest market share at 35.7 percent, followed by Oracle at 33.4 percent, and Microsoft at 17.7 percent. The overall market is very competitive with the major companies and smaller companies introducing many new features with each release.

TABLE 1.3
2003 Market Shares by Revenue of Enterprise Database Software[2]

Product	Total Market Share	Comments
Oracle 9i, 10g	39.9%	Dominates the Unix environment; strong performance in the Windows market also
IBM DB2, Informix	31.3%	Dominates the MVS and AS/400 environments; acquired Informix in 2001; 25% share of the Unix market
Microsoft SQL Server	12.1%	Dominate share of the Windows market; no presence in other environments
Other	16.7%	Includes Sybase, NCR Terradata, Progress Software, MySQL, PostgreSQL, open source Ingres, Firebird, and others

[2] Market shares according to a 2004 study by the International Data Corporation.

Open source DBMS products have begun to challenge the commercial DBMS products at the low end of the enterprise DBMS market. Although source code for open source DBMS products is available without charge, most organizations purchase support contracts so the open source products are not free. Still, many organizations have reported cost savings using open source DBMS products, mostly for non-mission-critical systems. MySQL, first introduced in 1995, is the leader in the open source DBMS market. PostgreSQL and open source Ingres are mature open source DBMS products. Firebird is a new open source product that is gaining usage.

In the market for desktop database software, Microsoft Access dominates at least in part because of the dominance of Microsoft Office. Desktop database software is primarily sold as part of office productivity software. With Microsoft Office holding about 90 percent of the office productivity market, Access holds a comparable share of the desktop database software market. Other significant products in the desktop database software market are Paradox, Approach, FoxPro, and FileMaker Pro.

To provide coverage of both enterprise and desktop database software, this book provides significant coverage of Oracle and Microsoft Access. In addition, the emphasis on the SQL standard in Parts 2 and 5 provides database language coverage for the other major products.

Because of the potential growth of personal computing devices, most major DBMS vendors have now entered the embedded DBMS market. The embedded DBMS market is now shared by smaller software companies such as iAnywhere Solutions and Solid Information Technology along with enterprise DBMS vendors Oracle and IBM.

1.4 Architectures of Database Management Systems

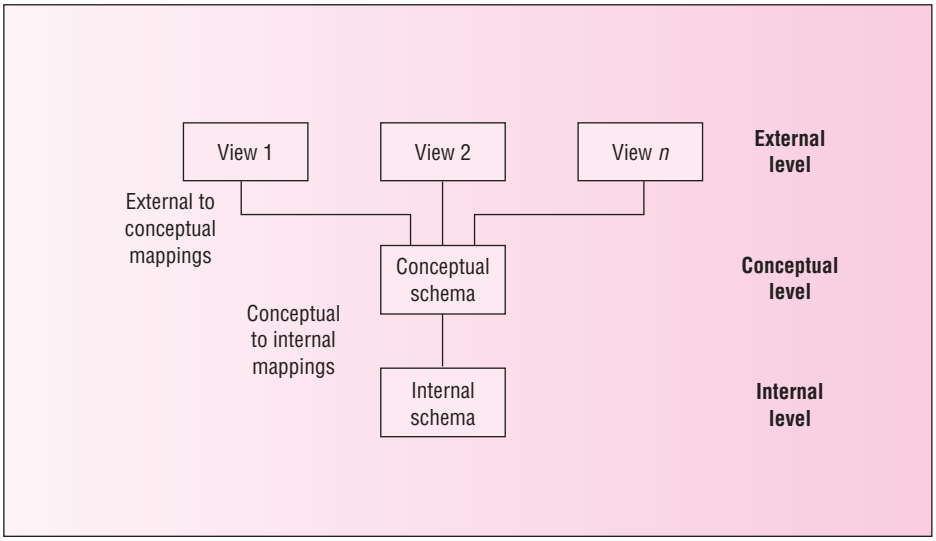
To provide insight about the internal organization of DBMSs, this section describes two architectures or organizing frameworks. The first architecture describes an organization of database definitions to reduce the cost of software maintenance. The second architecture describes an organization of data and software to support remote access. These architectures promote a conceptual understanding rather than indicate how an actual DBMS is organized.

1.4.1 Data Independence and the Three Schema Architecture

In early DBMSs, there was a close connection between a database and computer programs that accessed the database. Essentially, the DBMS was considered part of a programming language. As a result, the database definition was part of the computer programs that accessed the database. In addition, the conceptual meaning of a database was not separate from its physical implementation on magnetic disk. The definitions about the structure of a database and its physical implementation were mixed inside computer programs.

The close association between a database and related programs led to problems in software maintenance. Software maintenance encompassing requirement changes, corrections, and enhancements can consume a large fraction of software development budgets. In early DBMSs, most changes to the database definition caused changes to computer programs. In many cases, changes to computer programs involved detailed inspection of the code, a labor-intensive process. This code inspection work is similar to year 2000 compliance where date formats were changed to four digits. Performance tuning of a database was difficult because sometimes hundreds of computer programs had to be recompiled for every change. Because database definition changes are common, a large fraction of software maintenance resources were devoted to database changes. Some studies have estimated the percentage as high as 50 percent of software maintenance resources.

FIGURE 1.12
Three Schema Architecture



data independence
a database should have an identity separate from the applications (computer programs, forms, and reports) that use it. The separate identity allows the database definition to be changed without affecting related applications.

three schema architecture
an architecture for compartmentalizing database descriptions. The Three Schema Architecture was proposed as a way to achieve data independence.

The concept of data independence emerged to alleviate problems with program maintenance. Data independence means that a database should have an identity separate from the applications (computer programs, forms, and reports) that use it. The separate identity allows the database definition to be changed without affecting related applications. For example, if a new column is added to a table, applications not using the new column should not be affected. Likewise if a new table is added, only applications that need the new table should be affected. This separation should be even more pronounced if a change only affects physical implementation of a database. Database specialists should be free to experiment with performance tuning without concern about computer program changes.

In the mid-1970s, the concept of data independence led to the proposal of the Three Schema Architecture depicted in Figure 1.12. The word schema as applied to databases means database description. The Three Schema Architecture includes three levels of database description. The external level is the user level. Each group of users can have a separate external view (or view for short) of a database tailored to the group's specific needs.

In contrast, the conceptual and internal schemas represent the entire database. The conceptual schema defines the entities and relationships. For a business database, the conceptual schema can be quite large, perhaps hundreds of entity types and relationships. Like the conceptual schema, the internal schema represents the entire database. However, the internal schema represents the storage view of the database whereas the conceptual schema represents the logical meaning of the database. The internal schema defines files, collections of data on a storage device such as a hard disk. A file can store one or more entities described in the conceptual schema.

To make the three schema levels clearer, Table 1.4 shows differences among database definition at the three schema levels using examples from the features described in Section 1.2. Even in a simplified university database, the differences among the schema levels is clear. With a more complex database, the differences would be even more pronounced with many more views, a much larger conceptual schema, and a more complex internal schema.

The schema mappings describe how a schema at a higher level is derived from a schema at a lower level. For example, the external views in Table 1.4 are derived from the tables in the conceptual schema. The mapping provides the knowledge to convert a request using an external view (for example, HighGPAView) into a request using the tables in the conceptual

TABLE 1.4
University Database Example Depicting Differences among Schema Levels

Schema Level	Description
External	HighGPAView: data required for the query in Figure 1.7 FacultyAssignmentFormView: data required for the form in Figure 1.9 FacultyWorkLoadReportView: data required for the report in Figure 1.10
Conceptual	Student, Enrollment, Course, Faculty, and Enrollment tables and relationships (Figure 1.6)
Internal	Files needed to store the tables; extra files (indexed property in Figure 1.5) to improve performance

schema. The mapping between conceptual and internal levels shows how entities are stored in files.

DBMSs, using schemas and mappings, ensure data independence. Typically, applications access a database using a view. The DBMS converts an application's request into a request using the conceptual schema rather than the view. The DBMS then transforms the conceptual schema request into a request using the internal schema. Most changes to the conceptual or internal schema do not affect applications because applications do not use the lower schema levels. The DBMS, not the user, is responsible for using the mappings to make the transformations. For more details about mappings and transformations, Chapter 10 describes views and transformations between the external and conceptual levels. Chapter 8 describes query optimization, the process of converting a conceptual level query into an internal level representation.

The Three Schema Architecture is an official standard of the American National Standards Institute (ANSI). However, the specific details of the standard were never widely adopted. Rather, the standard serves as a guideline about how data independence can be achieved. The spirit of the Three Schema Architecture is widely implemented in third- and fourth-generation DBMSs.

1.4.2 Distributed Processing and the Client–Server Architecture

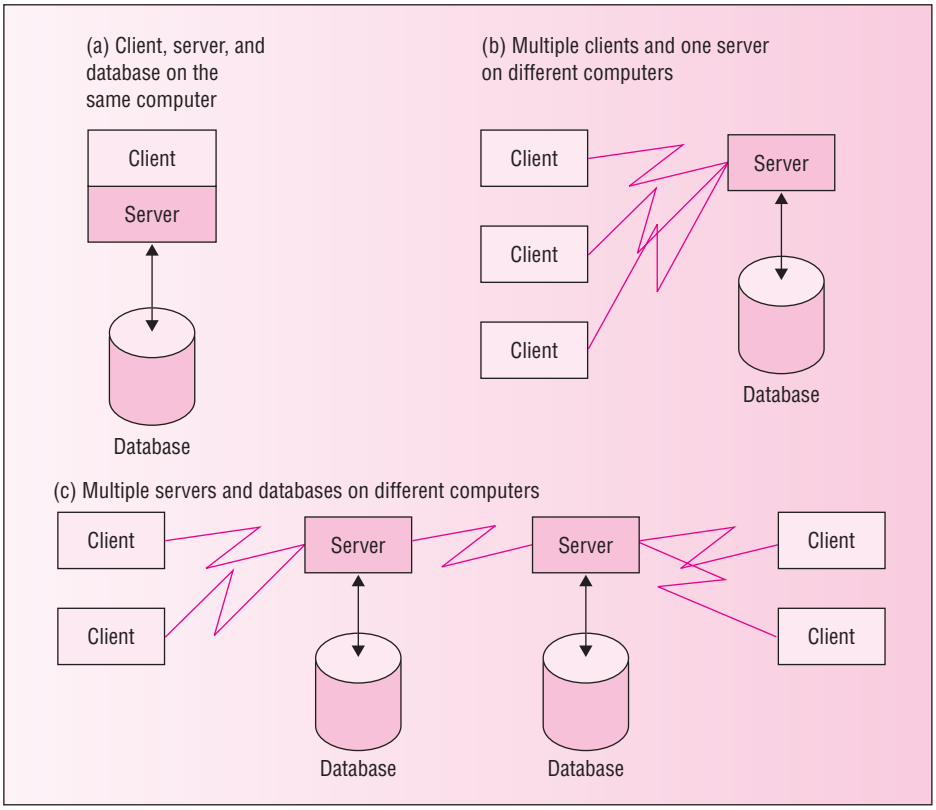
With the growing importance of network computing and the Internet, distributed processing is becoming a crucial function of DBMSs. Distributed processing allows geographically dispersed computers to cooperate when providing data access. A large part of electronic commerce on the Internet involves accessing and updating remote databases. Many databases in retail, banking, and security trading are now available through the Internet. DBMSs use available network capacity and local processing capabilities to provide efficient remote database access.

Many DBMSs support distributed processing using a client–server architecture. A client is a program that submits requests to a server. A server processes requests on behalf of a client. For example, a client may request a server to retrieve product data. The server locates the data and sends them back to the client. The client may perform additional processing on the data before displaying the results to the user. As another example, a client submits a completed order to a server. The server validates the order, updates a database, and sends an acknowledgement to the client. The client informs the user that the order has been processed.

client–server architecture
an arrangement of components (clients and servers) and data among computers connected by a network. The client–server architecture supports efficient processing of messages (requests for service) between clients and servers.

To improve performance and availability of data, the client–server architecture supports many ways to distribute software and data in a computer network. The simplest scheme is just to place both software and data on the same computer (Figure 1.13(a)). To take advantage of a network, both software and data can be distributed. In Figure 1.13(b), the server software and database are located on a remote computer. In Figure 1.13(c), the server software and the database are located on multiple remote computers.

FIGURE 1.13
Typical Client–Server Arrangements of Database and Software



The DBMS has a number of responsibilities in a client–server architecture. The DBMS provides software that can execute on both the client and the server. The client software is typically responsible for accepting user input, displaying results, and performing some processing of data. The server software validates client requests, locates remote databases, updates remote databases (if needed), and sends the data in a format that the client understands.

Client–server architectures provide a flexible way for DBMSs to interact with computer networks. The distribution of work among clients and servers and the possible choices to locate data and software are much more complex than described here. You will learn more details about client–server architectures in Chapter 17.

1.5 Organizational Impacts of Database Technology

This section completes your introduction to database technology by discussing the effects of database technology on organizations. The first section describes possible interactions that you may have with a database in an organization. The second section describes information resource management, an effort to control the data produced and used by an organization. Special attention is given to management roles that you can play as part of an effort to control information resources. Chapter 14 provides more detail about the tools and processes used in these management roles.

1.5.1 Interacting with Databases

Because databases are pervasive, there are a variety of ways in which you may interact with databases. The classification in Figure 1.14 distinguishes between functional users who interact with databases as part of their work and information systems professionals who

FIGURE 1.14
Classification of Roles

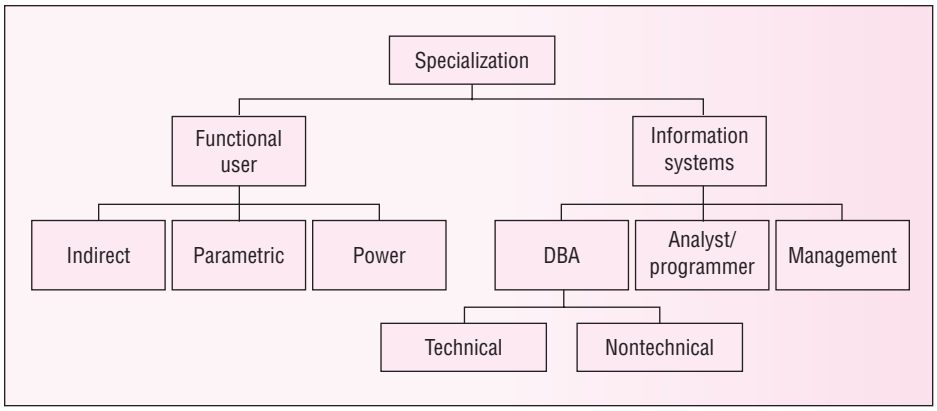


TABLE 1.5
Responsibilities of the Database Administrator

Technical	Nontechnical
Designing conceptual schemas	Setting database standards
Designing internal schemas	Devising training materials
Monitoring database performance	Promoting benefits of databases
Selecting and evaluating database software	Consulting with users
Designing client–server databases	
Troubleshooting database problems	

participate in designing and implementing databases. Each box in the hierarchy represents a role that you may play. You may simultaneously play more than one role. For example, a functional user in a job such as a financial analyst may play all three roles in different databases. In some organizations, the distinction between functional users and information systems professionals is blurred. In these organizations, functional users may participate in designing and using databases.

Functional users can play a passive or an active role when interacting with databases. Indirect usage of a database is a passive role. An indirect user is given a report or some data extracted from a database. A parametric user is more active than an indirect user. A parametric user requests existing forms or reports using parameters, input values that change from usage to usage. For example, a parameter may indicate a date range, sales territory, or department name. The power user is the most active. Because decision-making needs can be difficult to predict, ad hoc or unplanned usage of a database is important. A power user is skilled enough to build a form or report when needed. Power users should have a good understanding of nonprocedural access, a skill described in Parts 2 and 5 of this book.

Information systems professionals interact with databases as part of developing an information system. Analyst/programmers are responsible for collecting requirements, designing applications, and implementing information systems. They create and use external views to develop forms, reports, and other parts of an information system. Management has an oversight role in the development of databases and information systems.

database administrator
a support position that specializes in managing individual databases and DBMSs.

Database administrators assist both information systems professionals and functional users. Database administrators have a variety of both technical and nontechnical responsibilities (Table 1.5). Technical skills are more detail-oriented; nontechnical responsibilities are more people-oriented. The primary technical responsibility is database design. On the nontechnical side, the database administrator's time is split among a number of activities. Database administrators can also have responsibilities in planning databases and evaluating DBMSs.

1.5.2 Information Resource Management

Information resource management is a response to the challenge of effectively utilizing information technology. The goal of information resource management is to use information technology as a tool for processing, distributing, and integrating information throughout an organization. Management of information resources has many similarities with managing physical resources such as inventory. Inventory management involves activities such as safeguarding inventory from theft and deterioration, storing it for efficient usage, choosing suppliers, handling waste, coordinating movement, and reducing holding costs. Information resource management involves similar activities: planning databases, acquiring data, protecting data from unauthorized access, ensuring reliability, coordinating flow among information systems, and eliminating duplication.

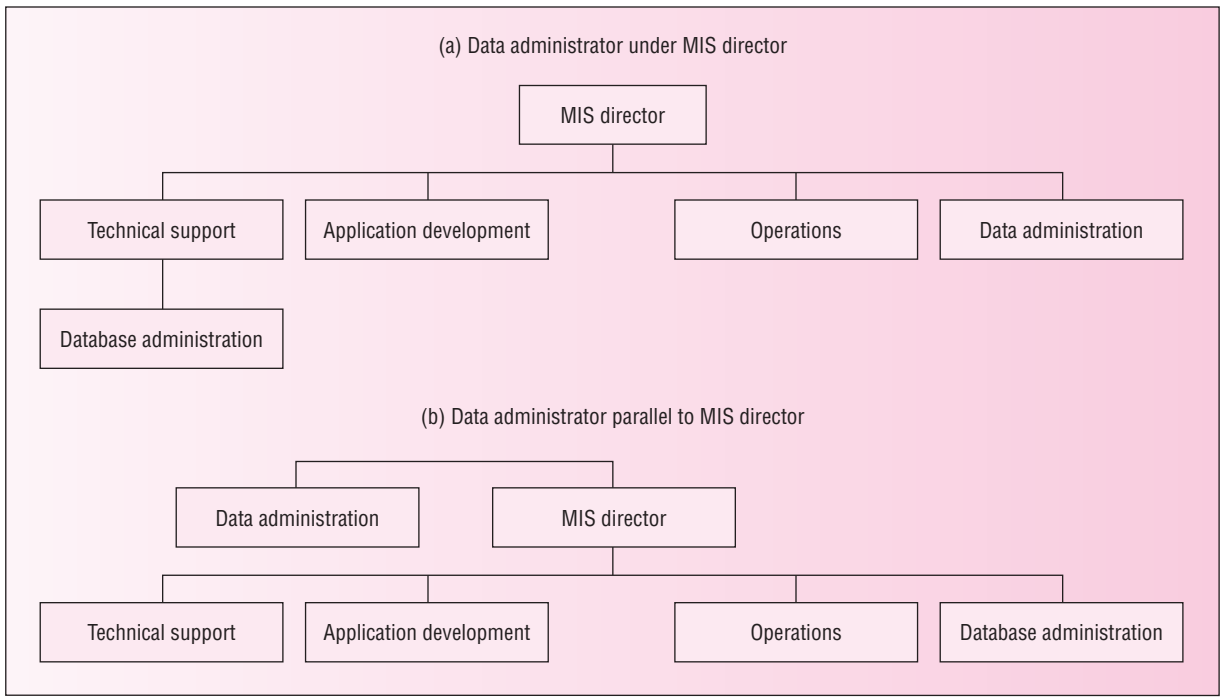
data administrator
a management position that performs planning and policy setting for the information resources of an entire organization.

As part of controlling information resources, new management responsibilities have arisen. The data administrator is a management role with many of these responsibilities; the major responsibility being planning the development of new databases. The data administrator maintains an enterprise data architecture that describes existing databases and new databases and also evaluates new information technologies and determines standards for managing databases.

The data administrator typically has broader responsibilities than the database administrator. The data administrator has primarily a planning role, while the database administrator has a more technical role focused on individual databases and DBMSs. The data administrator also views the information resource in a broader context and considers all kinds of data, both computerized and noncomputerized. A major effort in many organizations is to computerize nontraditional data such as video, training materials, images, and correspondence. The data administrator develops long-range plans for nontraditional data, while the database administrator implements the plans using appropriate database technology.

Because of broader responsibilities, the data administrator typically is higher in an organization chart. Figure 1.15 depicts two possible placements of data administrators and

FIGURE 1.15 **Organizational Placement of Data and Database Administration**



database administrators. In a small organization, both roles may be combined in systems administration.

Closing Thoughts

Chapter 1 has provided a broad introduction to DBMSs. You should use this background as a context for the skills you will acquire in subsequent chapters. You learned that databases contain interrelated data that can be shared across multiple parts of an organization. DBMSs support transformation of data for decision making. To support this transformation, database technology has evolved from simple file access to powerful systems that support database definition, nonprocedural access, application development, transaction processing, and performance tuning. Nonprocedural access is the most vital element because it allows access without detailed coding. You learned about two architectures that provide organizing principles for DBMSs. The Three Schema Architecture supports data independence, an important concept for reducing the cost of software maintenance. Client–server architectures allow databases to be accessed over computer networks, a feature vital in today's networked world.

The skills emphasized in later chapters should enable you to work as an active functional user or analyst. Both kinds of users need to understand the skills taught in the second part of this book. The fifth part of the book provides skills for analysts/programmers. This book also provides the foundation of skills to obtain a specialist position as a database or data administrator. The skills in the third, fourth, sixth, and seventh parts of this book are most useful for a position as a database administrator. However, you will probably need to take additional courses, learn details of popular DBMSs, and acquire management experience before obtaining a specialist role. A position as a database specialist can be an exciting and lucrative career opportunity that you should consider.

Review Concepts

- Database characteristics: persistent, interrelated, and shared.
- Features of database management systems (DBMSs).
- Nonprocedural access: a key to software productivity.
- Transaction: a unit of work that should be processed reliably.
- Application development using nonprocedural access to specify the data requirements of forms and reports.
- Procedural language interface for combining nonprocedural access with a programming language such as COBOL or Visual Basic.
- Evolution of database software over four generations of technological improvement.
- Current emphasis on database software for multimedia support, distributed processing, more powerful operators, and data warehouses.
- Types of DBMSs: enterprise, desktop, embedded.
- Data independence to alleviate problems with maintenance of computer programs.
- Three Schema Architecture for reducing the impact of database definition changes.
- Client–server architecture for using databases over computer networks.
- Database specialist roles: database administrator and data administrator.
- Information resource management for utilizing information technology.

Questions

1. Describe a database that you have used on a job or as a consumer. List the entities and relationships that the database contains. If you are not sure, imagine the entities and relationships that are contained in the database.
2. For the database in question 1, list different user groups that can use the database.
3. For one of the groups in question 2, describe an application (form or report) that the group uses.
4. Explain the persistent property for databases.
5. Explain the interrelated property for databases.
6. Explain the shared property for databases.
7. What is a DBMS?
8. What is SQL?
9. Describe the difference between a procedural and a nonprocedural language. What statements belong in a procedural language but not in a nonprocedural language?
10. Why is nonprocedural access an important feature of DBMSs?
11. What is the connection between nonprocedural access and application (form or report) development? Can nonprocedural access be used in application development?
12. What is the difference between a form and a report?
13. What is a procedural language interface?
14. What is a transaction?
15. What features does a DBMS provide to support transaction processing?
16. For the database in question 1, describe a transaction that uses the database. How often do you think that the transaction is submitted to the database? How many users submit transactions at the same time? Make guesses for the last two parts if you are unsure.
17. What is an enterprise DBMS?
18. What is a desktop DBMS?
19. What is an embedded DBMS?
20. What were the prominent features of first-generation DBMSs?
21. What were the prominent features of second-generation DBMSs?
22. What were the prominent features of third-generation DBMSs?
23. What are the prominent features of fourth-generation DBMSs?
24. For the database you described in question 1, make a table to depict differences among schema levels. Use Table 1.4 as a guide.
25. What is the purpose of the mappings in the Three Schema Architecture? Is the user or DBMS responsible for using the mappings?
26. Explain how the Three Schema Architecture supports data independence.
27. In a client–server architecture, why are processing capabilities divided between a client and server? In other words, why not have the server do all the processing?
28. In a client–server architecture, why are data sometimes stored on several computers rather than on a single computer?
29. For the database in question 1, describe how functional users may interact with the database. Try to identify indirect, parametric, and power uses of the database.
30. Explain the differences in responsibilities between an active functional user of a database and an analyst. What schema level is used by both kinds of users?
31. Which role, database administrator or data administrator, is more appealing to you as a long-term career goal? Briefly explain your preference.
32. What market niche is occupied by open source DBMS products?

Problems

Because of the introductory nature of this chapter, there are no problems in this chapter. Problems appear at the end of most other chapters.

References for Further Study

The *DBAZine* (www.dbazine.com), the *Intelligent Enterprise* magazine (www.iemagazine.com), and the *Advisor.com* (www.advisor.com) websites provide detailed technical information about commercial DBMSs, database design, and database application development. To learn more about the role of database specialists and information resource management, you should consult Mullin (2002).