chapter

1

Introduction to Database Management

Learning Objectives

This chapter introduces 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
- Explain conceptual architectures for databases used in distributed processing and software maintenance
- Perceive career opportunities related to database development and database administration

OVERVIEW

You may not be aware of it, but database technology dramatically affects your life. Modern organizations cannot operate efficiently without databases and associated database technology. You encounter databases daily through activities such as shopping at a supermarket, withdrawing cash using an automated teller machine, making an airline reservation, ordering a book online, and registering for classes. The proliferation of databases and supporting database technology provides convenience in your daily life.

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 such as consumer preferences, telecommunications usage, credit history, television viewing habits, transportation usage, and spending patterns. Database technology helps 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. The government uses information mined from databases to target taxation enforcement, refine pollution control efforts, target interest groups for election appeals, and develop new laws.

In Part 1, Chapter 1 provides a starting point for your exploration of database technology surveyingdatabase characteristics, database management system features, system architectures, and human roles in managing and using databases. Chapter 1 also provides a broad picture of database technology and shares the excitement about the journey ahead. Chapter 2 provides a conceptual overview of the database development process.

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 types of simple facts and nonconventional facts such as medical images, customer reviews, fingerprints, product photos, and maps. With the proliferation of the Internet and the means to capture data in digital format, 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 these points.

- Persistent means that data reside on stable storage such as a magnetic disk and solid-state devices. For example, organizations retain data about customers, suppliers, and inventory on stable storage because of the need to reference these details repetitively. A variable in a web page is not persistent because it resides in the main memory and disappears after terminating the visit to the web page. 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 the 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 health savings account. Because collecting, storing, and maintaining data is costly, only data likely to be relevant for actions and 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 simultaneously trying to change the same data, 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, etc.) to order data (order number, order date, etc.) 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 the salary history of employees. A typical business database may have hundreds of types of entities and relationships.

To depict these characteristics, let us consider several 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 university processes for registering for classes, assigning faculty to course offerings, recording grades, and scheduling courses. Relationships in the university database support answers to questions such as

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

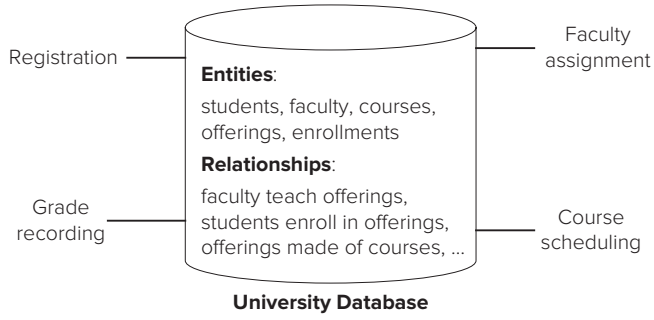


FIGURE 1.1
Depiction of a Simplified University Database

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

Next, let us consider a water utility database as depicted in Figure 1.2. The water utility database supports billing customers for water usage. Periodically, the water utility measures a customer's water consumption from a meter and generates a bill. 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 their last payment?

Finally, let us consider a hospital database as depicted in Figure 1.3. The hospital database supports the treatment of patients by health care providers. Many different health care providers read and contribute to a patient's medical record. Physicians make diagnoses and prescribe treatments based on symptoms. Nurses monitor symptoms and provide medication. Dietary professionals design meal plans according to dietary restrictions. 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 physician make for a patient?

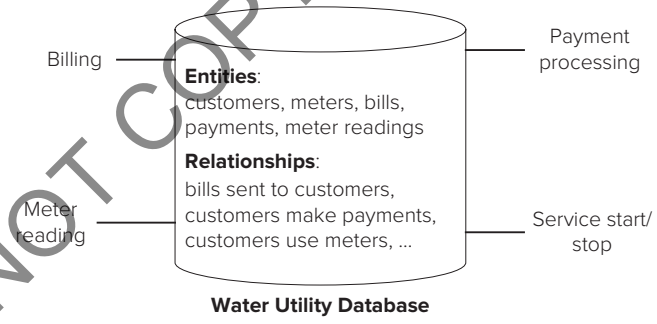


FIGURE 1.2
Depiction of a Simplified Water Utility Database

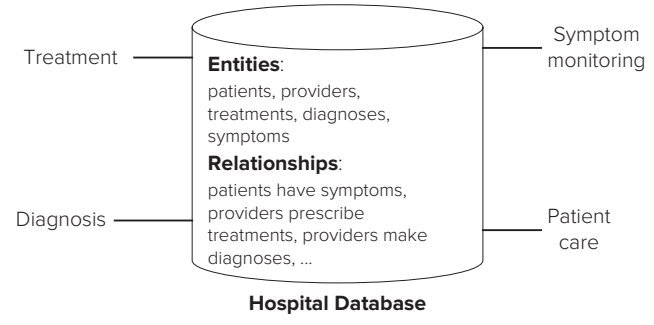


FIGURE 1.3
Depiction of a Simplified Hospital Database

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, 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 types of 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 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. Most examples are from Oracle, a prominent commercial DBMS, and PostgreSQL, a prominent open-source 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, a database designer specifies entities and relationships. 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 entity types, relationships, integrity constraints, and authorization rights
Nonprocedural access	Language and graphical tools to access data without complicated coding
Procedural language interface	Language that combines nonprocedural access with full capabilities of a programming language such as Java or Javascript
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 Rows in the Student Table

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

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 the SQL CREATE TABLE statement to define tables and relationships.

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

In addition to SQL, many DBMSs provide client tools for graphically defining and displaying database designs. Client tools interact with a database server to process SQL CREATE TABLE statements. Figures 1.5 and 1.6 depict graphical tools for defining tables and displaying database diagrams with tables and relationships. The table properties window in the pgAdmin client for PostgreSQL provides convenient editing of table definitions. Using the properties window in Figure 1.5, a user can define properties of columns such as the data type, length, and scale. The Data Modeling tool for Oracle provides graphical display of database diagrams with tables and relationships. After defining the structure, a database can be populated. The data in Figure 1.4 should be added after defining details of tables and relationships.

1.2.2 Nonprocedural Access

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 with large balances or 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 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.

Nonprocedural access can reduce the number of lines of code by a factor of 100 compared to procedural access. Because a large part of business software involves data access, nonprocedural access can dramatically improve software productivity.

To appreciate the significance of nonprocedural access, consider an analogy to planning a vacation. You specify the 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 a plan with details about the best route to your destination, the most desirable hotel, ground transportation, and so on. A planning professional can facilitate your planning process by completing these details. Like a planning

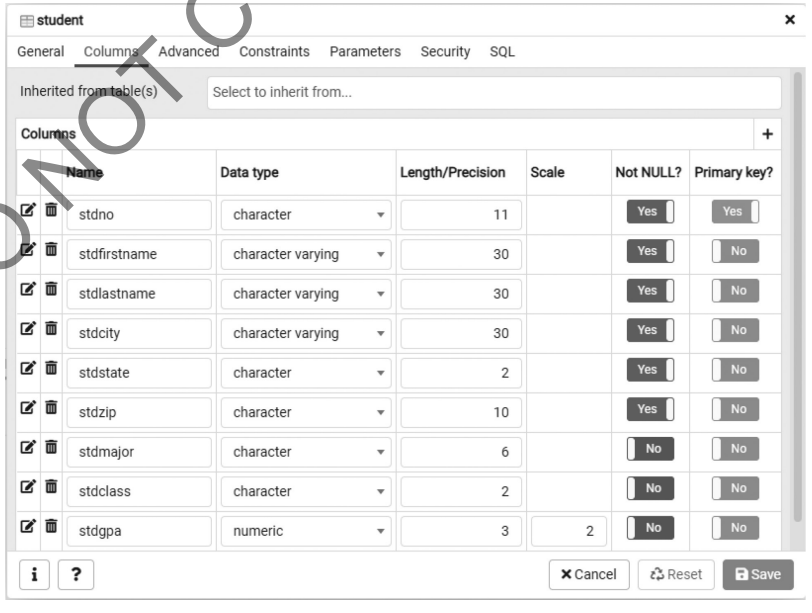


FIGURE 1.5
Table Properties Window in the pgAdmin Client for PostgreSQL

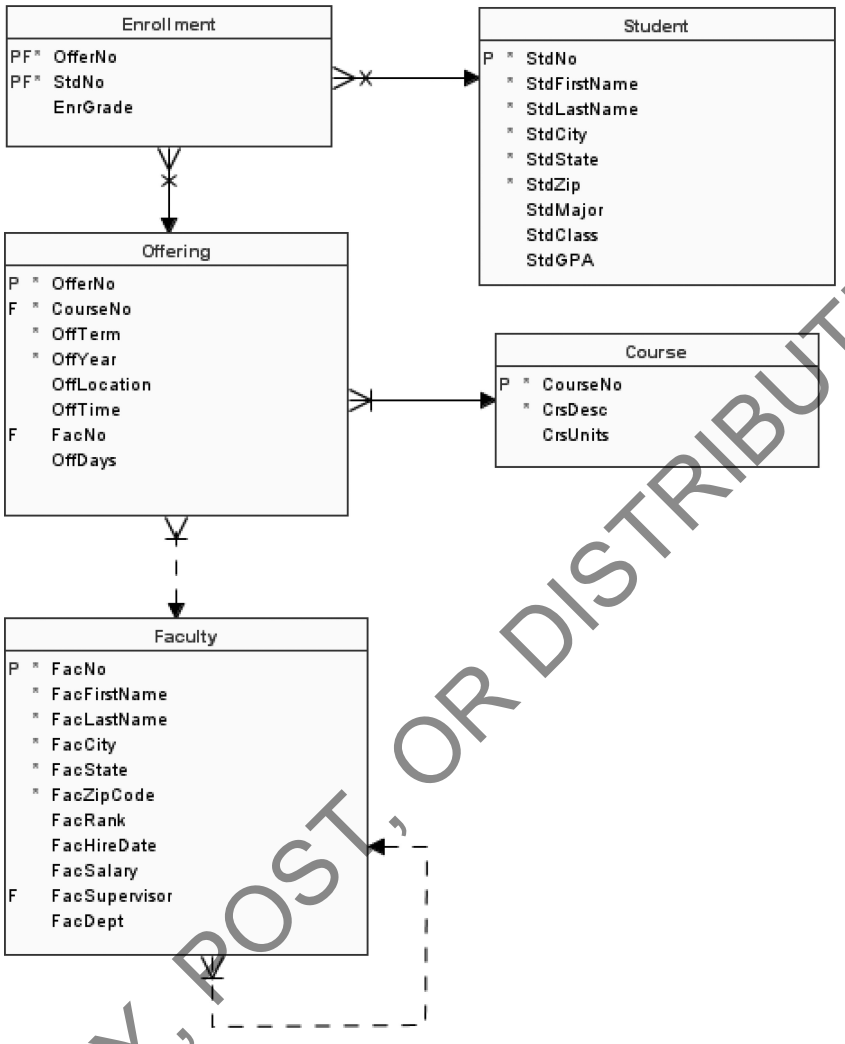


FIGURE 1.6
Database Diagram in the Oracle Data Modeler

professional, a DBMS performs detailed planning to answer queries expressed in a nonprocedural language.

The standard tool for nonprocedural access is the SQL SELECT statement. Most DBMSs provide an SQL client to enter and execute SQL statements, including the SELECT statement. Figure 1.7 depicts the execution of an SQL SELECT statement using the Query Tool of the pgAdmin client for PostgreSQL. The Query Editor tab supports statement entry with some editing assistance for keywords shown in magenta. The results appear below the Query Editor window in the Data Output tab after selecting the execute button ▶.

Most DBMSs, as well as third-party vendors, also provide graphical tools to access databases. Figure 1.8 depicts the Query Builder tool available in the Oracle SQL Developer. To specify a query, a user indicates the tables, relationships, and columns. Figure 1.9 shows the result of executing the graphical specification in Figure 1.8.

Nonprocedural access can also be used in graphical tools for building applications with forms and reports. Data entry forms support convenient data entry and display, while reports enhance the appearance of data displays. 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 or

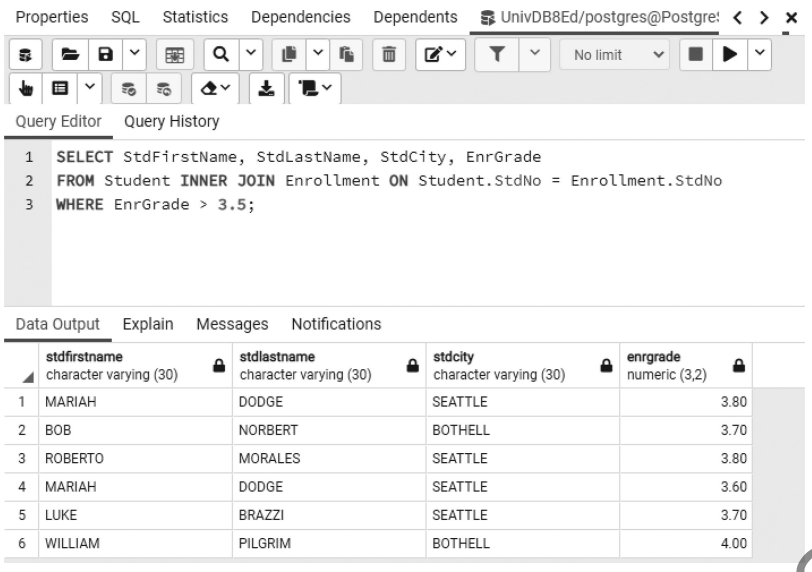


FIGURE 1.7
pgAdmin SQL Client with SELECT Statement Execution

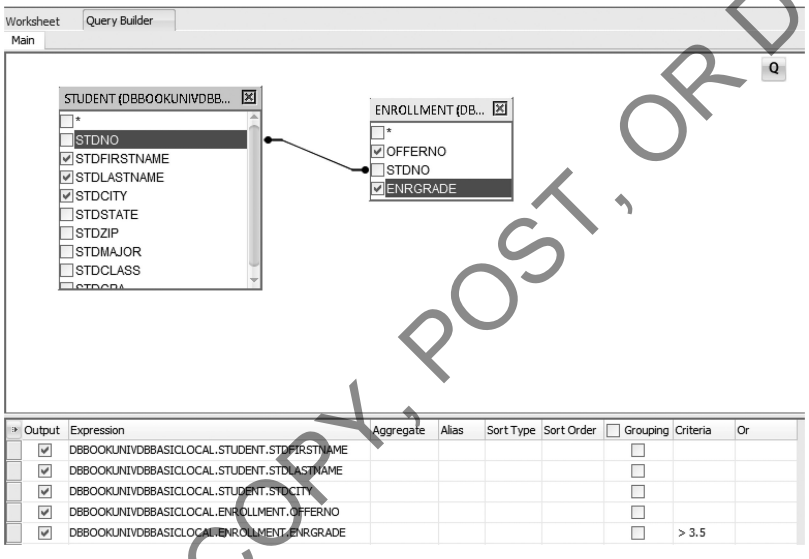


FIGURE 1.8
Query Builder Window in the Oracle SQL Developer

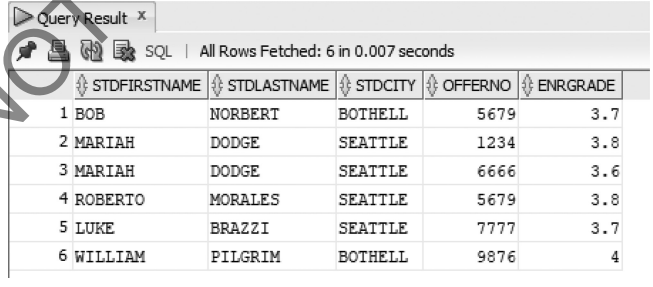


FIGURE 1.9
Result of Executing Query in Figure 1.8

graphical tool. To complete a form or report definition, the user indicates the formatting of data, user interaction, and other details.

1.2.3 Procedural Language Interface

Nonprocedural access, although convenient and powerful, does not support the development of data-intensive applications such as shopping cart web pages for order entry

and data mining algorithms searching large databases for hidden patterns. To develop complex applications with database access, DBMSs provide the full capabilities of a programming language along with embedded nonprocedural access. A **procedural language interface** combines the full capabilities of a computer programming language with nonprocedural access using SQL.

Procedural Language Interface
a method to combine a nonprocedural language such as SQL with a programming language such as Java or Visual Basic.

DBMSs support two language styles for integrating a procedural language with SQL. A statement-level interface involves a new language combining procedural statements and SQL statements. For example, Oracle provides the database programming language PL/SQL, PostgreSQL features PL/pgSQL, and Microsoft SQL Server supports Transact-SQL. Chapter 11 describes procedural language interfaces and the Oracle PL/SQL language. The statement-level interface has statements to establish database connections, execute SQL statements, use the results of an SQL statement, associate programming variables with database columns, and handle exceptions in SQL statements.

The call-level interface, the second language style, contains procedures and data definitions to combine the results of SQL statements with programming language statements. In a call-level interface, the statements of the host programming language are not extended. DBMSs provide call-level interfaces for many programming languages such Java, Java-script, Visual Basic, and C++. The two most popular call-level interfaces are the Open Database Connectivity supported by Microsoft and the Java Database Connectivity supported by Oracle. These call-level interfaces provide procedures to establish database connections, execute SQL statements, use the results of an SQL statement, associate programming variables with database columns and handle exceptions in SQL statements.

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 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 an unseen, back-office affair. The user does not know the details about transaction processing other than the assurances about reliability.

Database tuning involves components to monitor and improve performance. Some DBMSs can monitor database performance and generate events indicating conditions that may warrant investigation. DBMSs provide components to improve performance, such as reorganization of a database, selection of physical structures, and repair of 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, known as enterprise DBMSs, support databases 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 or a smartphone. Embedded DBMSs provide limited transaction processing features but have low memory, processing, and storage requirements.

1.2.5 Third-Party Tools

Third-party tools extend features directly provided by DBMSs, especially important for organizations using multiple DBMSs. Third-party SQL clients provide flexibility if an organization needs to execute SQL statements using different DBMSs. Third-party

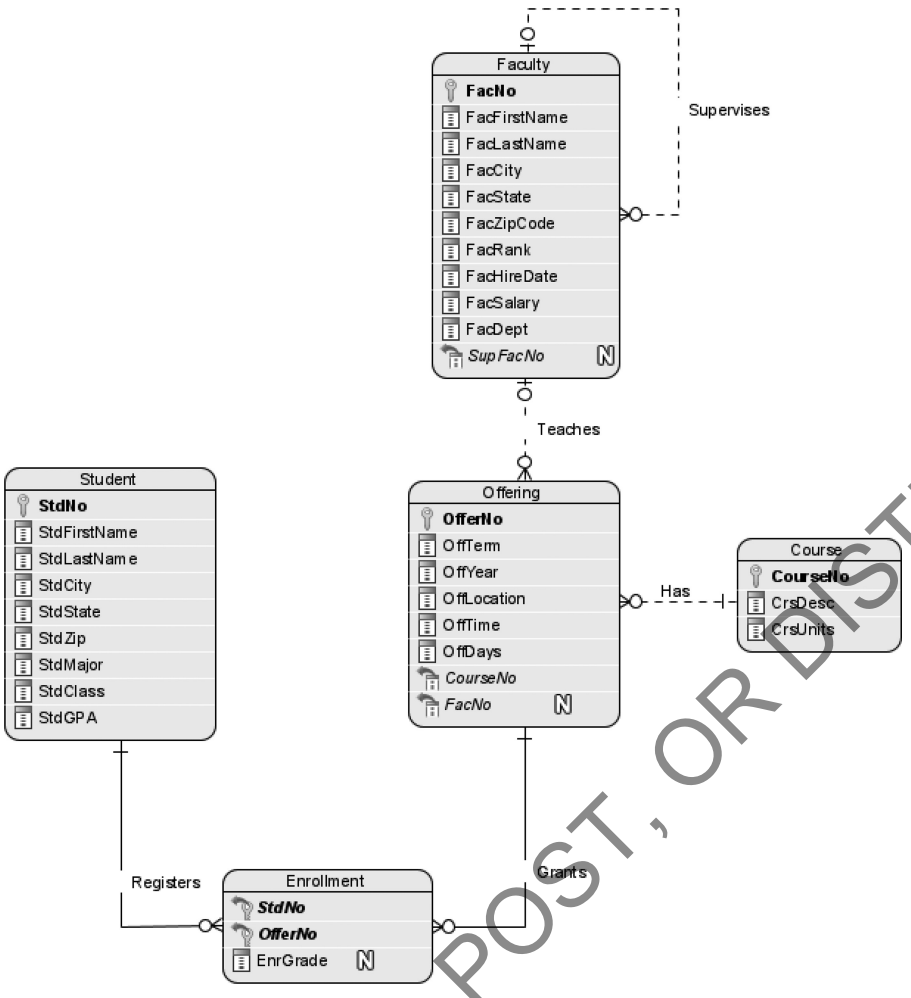


FIGURE 1.10
Entity Relationship Diagram for the University Database using Visual Paradigm

design tools extend database definition and tuning capabilities provided by DBMSs. Figure 1.10 shows a database diagram (an entity-relationship diagram) created with Visual Paradigm, a tool for database design and software development. Visual Paradigm provides features to convert a database design into tables compliant with the SQL standard supported by most commercial DBMSs.

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 a context to appreciate today's DBMSs, this section reviews 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 computer programs with procedural code were necessary for data retrieval. For

[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.

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 2010s	4th generation	Object	Multimedia, active, distributed processing, more powerful operators, data warehouse processing, XML enabled, cloud computing, big data demands, semi-structured data

example, a program could be written to retrieve all customer records or just to 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 type rather than many entity types and relationships managed by a DBMS.

The second-generation products were the first true DBMSs because they could manage multiple entity types and relationships. However, to access data, a computer program still had to be written. Second-generation systems are called "navigational" because the programmer had to write code to search among a network of linked records. Some 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 universities 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 a large improvement over navigational access, third-generation systems supplanted the second generation. Since the technology was so different, most new systems were founded by start-up companies rather than by vendors of previous generation products. IBM was the major exception. IBM's weight led to the adoption of SQL as a widely accepted standard.

Fourth-generation DBMSs have extended the boundaries of database technology to unconventional data, new kinds of distributed processing, data warehouse processing, and big data demands, especially with semi-structured data. As an early emphasis, fourth-generation DBMSs supported unconventional data types such as images, videos, maps, sounds, animations, and web pages. Most DBMSs now feature convenient ways to publish static and dynamic web pages using the eXtensible Markup Language (XML) as a publishing standard. Because these DBMSs view any kind of data as an object to manage, fourth-generation systems were called object-relational.

In the last 20 years, DBMS vendors have extended their fourth-generation products 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 dominates data warehouse processing, whereas a mixture of updating and retrieving data occur for databases that support the daily operations of an organization. Part 6 covers data warehouse concepts and DBMS features to support data warehouse processing.

Cloud computing is a recent area of product development for both established DBMS vendors and new vendors. Cloud computing supports on-demand and pay-per-per-use access for both data and software. Cloud computing usage is web-based without fixed costs of software ownership. Major DBMS vendors have developed cloud computing models as an alternative to their traditional approach of product licensing and ownership. In addition, new vendors have created DBMS products tailored to the cloud computing model.

Part of the promise of cloud computing is support for applications with exploding data growth known as big data. The growth in data comes from a variety of sources such as sensors in smartphones, energy meters, and automobiles, the interaction of individuals in social media websites, radio frequency identification tags in retail, and digitized media content in medicine, entertainment, and security. Big data exceeds the limits of commercial database software to support applications with exploding data growth.

NoSQL (Not only SQL) database technology has been developed to deal with some of the challenges of big data. As the name implies, NoSQL database technology does not use the traditional relational database model and SQL standard. Instead, NoSQL database products use simplified database models, less stringent transaction processing models, and distributed processing to reduce bottlenecks for dealing with big data. NoSQL products cover a wide range of data models to support the management of semi-structured data with key-record pairs, documents, and graphs.

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 often developed as open-source software with subscriptions for premium services. The existing companies seem to have the upper hand, but the open-source DBMS products have gained important commercial usage. This textbook provides balanced coverage between the industry-leading enterprise DBMS, Oracle, and the most prominent open-source DBMS, PostgreSQL.

1.3.2 Popularity of DBMSs

DB-Engines.com ranks DBMS products by popularity using the number of mentions on websites, frequency of search in Google Trends, job offers in leading job websites, and profiles in professional websites. The DB-Engines ranking (top 10) in June 2021 of DBMSs supporting the SQL standard was Oracle, MySQL, Microsoft SQL Server, PostgreSQL, IBM DB2, SQLite, Microsoft Access, MariaDB, Hive, and Microsoft Azure SQL Database. The ranking combining SQL-compliant and non-SQL compliant DBMSs was Oracle, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB (NoSQL product), IBM DB2, Redis (NoSQL), Elasticsearch, SQLite, and Microsoft Access.

Open-source DBMS products have begun to challenge the commercial DBMS products in 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. In addition, some organizations offer a community edition without charge and an enterprise edition with traditional license fees. Still, many organizations have reported a lower cost of ownership using open-source DBMS products. MySQL, first introduced in 1995, is the leader in the open-source DBMS market. Open-source DBMS products feature prominently in the DB-Engines.com ranking with six open-source products (MySQL, PostgreSQL, MongoDB, MariaDB, Redis, and SQLite).

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% of the office productivity market, Access holds a comparable share of the desktop database software market. Other significant products in the desktop database market are open-source products LibreOffice Base and OpenOffice Base, along with commercial product FileMaker Pro.

Because of the potential growth of personal computing devices, most major DBMS vendors have now entered the embedded DBMS market. An embedded DBMS provides tight integration with application software. Thus, embedded DBMSs remain hidden from users with little or no maintenance. Embedded DBMS software is sold primarily by value-added software resellers as part of an application, such as an accounting package. Some of the leading embedded DBMS products are Oracle Berkeley DB, Firebird Embedded, MySQL Embedded, SQLite, Microsoft SQL Server Compact, IBM Informix Embedded, and SAP SQL Anywhere Embedded Database.

The market for cloud-based DBMSs is rapidly evolving so market shares and size are difficult to determine. Most major DBMS vendors offer cloud-based solutions with some vendors providing both traditional SQL and emerging NoSQL products. For example, Amazon offers Relational Data Service (SQL) and Amazon DynamoDB (NoSQL), Oracle Cloud provides many versions of Oracle Database, and Microsoft offers Azure (SQL) and DocumentDB (NoSQL). The impact of cloud computing on the DBMS market has begun to mature in 2021.

1.4 ARCHITECTURES OF DATABASE MANAGEMENT SYSTEMS

This section provides insight into the internal organization of DBMSs, by describing 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 actual DBMS implementation.

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 like year 2000 compliance in which date formats were changed to four digits. Performance tuning 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% of software maintenance resources.

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.

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 the physical implementation of a database. Database specialists should be free to experiment with performance tuning without concern about computer program changes.

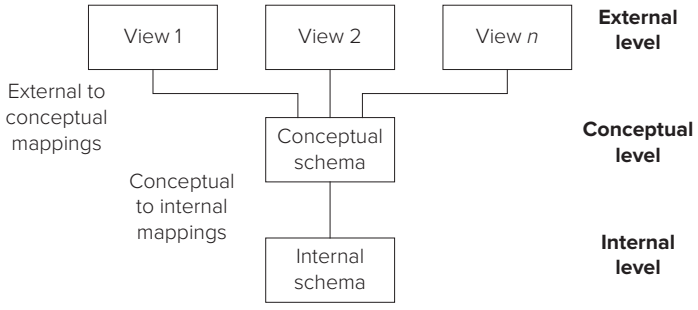


FIGURE 1.11
Three Schema Architecture

In the mid-1970s, the concept of data independence led to the proposal of the Three Schema Architecture depicted in Figure 1.11. 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.

Three Schema Architecture an architecture for compartmentalizing database descriptions. The Three Schema Architecture was proposed to support data independence.

In contrast, the conceptual and internal schemas represent the entire database. The conceptual schema defines the entity types 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 which are collections of data on a storage device such as a hard disk. A file can store one or more entity types described in the conceptual schema.

To make the three schema levels clearer, Table 1-3 shows differences among database definitions at the three schema levels. Even in a simplified university database, the differences among the schema levels are 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 1-3 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 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

TABLE 1-3
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 a Faculty Assignment form FacultyWorkLoadReportView: data required for a Faculty Workload report
Conceptual	Student, Enrollment, Course, Faculty, and Enrollment tables and relationships (Figure 1.6)
Internal	Data files needed to store the tables; index files to improve performance

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 data independence. The spirit of the Three Schema Architecture is widely implemented in third- and fourth-generation DBMSs.

1.4.2 Parallel and Distributed Database Processing

With the growing importance of computer networks and electronic commerce, 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 involves accessing and updating remote databases. Many databases in retail, banking, and security trading are now available through electronic commerce websites. DBMSs use available network capacity and local processing capabilities to provide efficient remote database access.

Distributed processing can be applied to databases to distribute tasks among servers, divide a task among processing resources, and distribute data among network sites. To distribute tasks among servers, many DBMSs use the 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. DBMSs may employ one or more levels of servers to distribute different kinds of database processing. In Figure 1.12(a), the database server and database are located on a remote computer. In Figure 1.12(b), an additional middleware server is added to efficiently process messages from many clients.

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

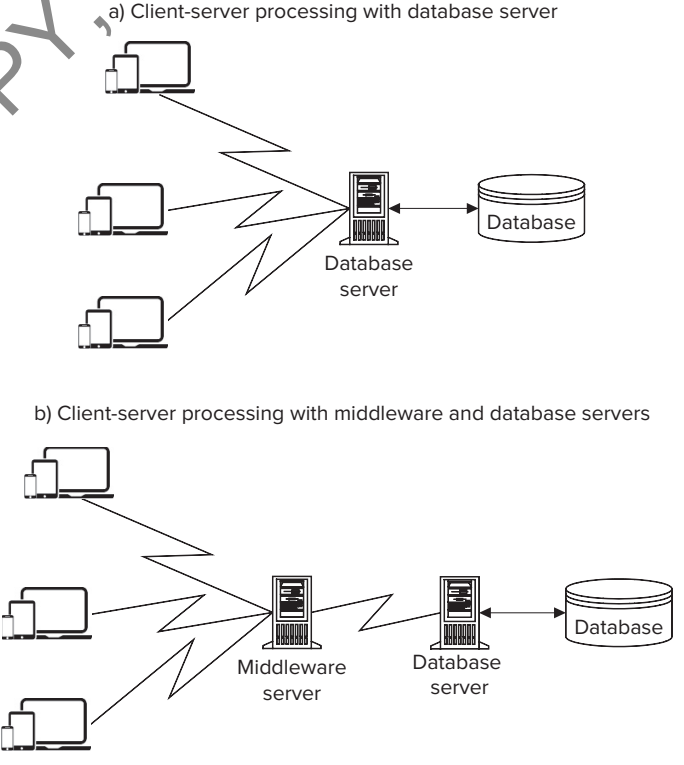


FIGURE 1.12 Typical Client-Server Architectures

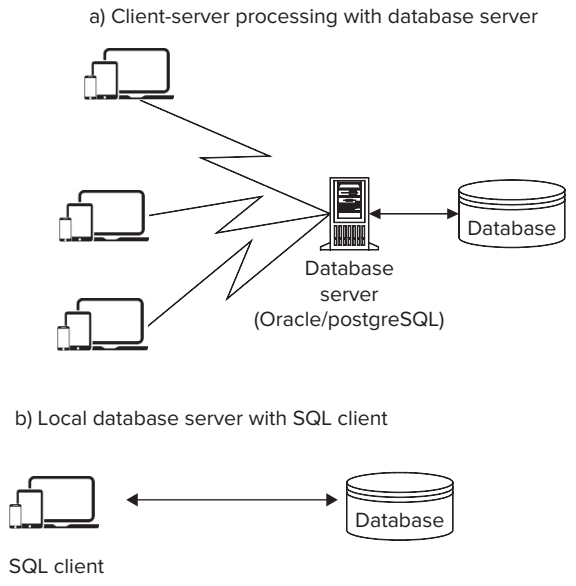


FIGURE 1.13
Client-Server Architecture for Course Usage

For usage of this textbook in a classroom environment, Figure 1.13 depicts a client-server architecture with a database server and an SQL client. In Figure 1.13a, the database server (Oracle or PostgreSQL) and SQL client (Oracle SQL Developer or PostgreSQL pgAdmin) reside on different computers supporting a classroom environment with students sharing the same database server. The database server receives SQL statements from the SQL client and returns statement results (usually rows) to the SQL client. Alternatively, a local database server and SQL client may reside on the same computer (Figure 1.13b) if an external server is unavailable. When residing in the same local computer, the database server and SQL client execute as separate processes. Both Oracle and PostgreSQL support remote and local database servers.

In the last decade, parallel database technology has gained commercial acceptance for large organizations. Most enterprise DBMS vendors and some open-source DBMSs support parallel database technology to meet market demand. Organizations are utilizing these products to realize the benefits of improved performance and availability. Parallel database processing can improve performance through speedup (performing a task faster) and scaleup (performing more work simultaneously). Parallel database processing can increase availability because a DBMS can dynamically adjust to the level of available resources. Figure 1.14 depicts two common parallel database architectures that can provide improved performance and availability. In Figure 1.14(a) known as the shared disk (SD) architecture, each processor has its own memory, but the processors share the disks. In Figure 1.14(b) known as shared nothing (SN) architecture, each processor has its own memory and disks.

Parallel DBMS
a DBMS capable of utilizing tightly-coupled computing resources (processors, disks, and memory). Tight coupling is achieved by networks with data exchange time comparable to the time of the data exchange with a disk. Parallel database technology promises performance improvements and high availability.

Distributed data provides local control and reduced communication costs. Distributing a database allows the location of data to match an organization's structure. Decisions about sharing and maintaining data can be set locally to provide control closer to the data usage. Data should be located so that 80 percent of the requests are local. Local requests incur little communication costs and delays compared to remote requests. Figure 1.15 depicts a distributed database with three sites in Denver, London, and Tokyo. Each site can control access to its local data and cooperate to provide data sharing for tasks needing data from more than one site.

Distributed Database
a database in which parts are located at different network sites. Distributed database technology supports local control of data, data sharing for requests involving data from more than one site, and reduced communication overhead.

Client-server architectures, parallel database processing, and distributed databases provide flexible ways for DBMSs to interact with computer networks. The distribution of data and processing 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 these architectures in Chapter 18.

FIGURE 1.14
Basic Parallel Database Architectures

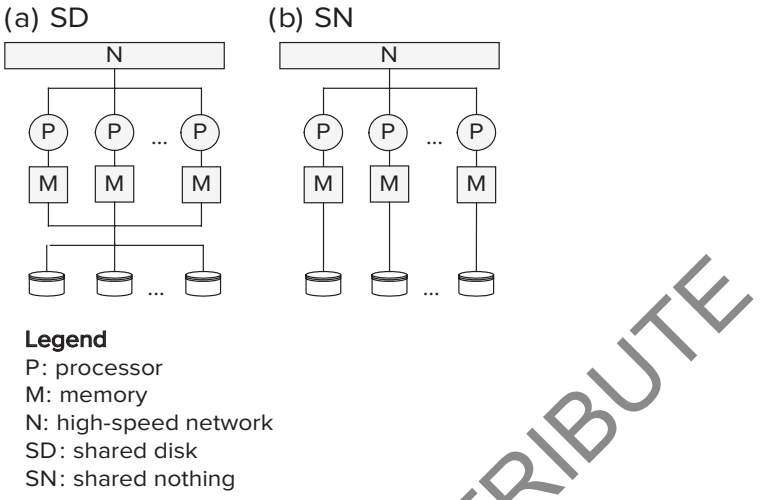
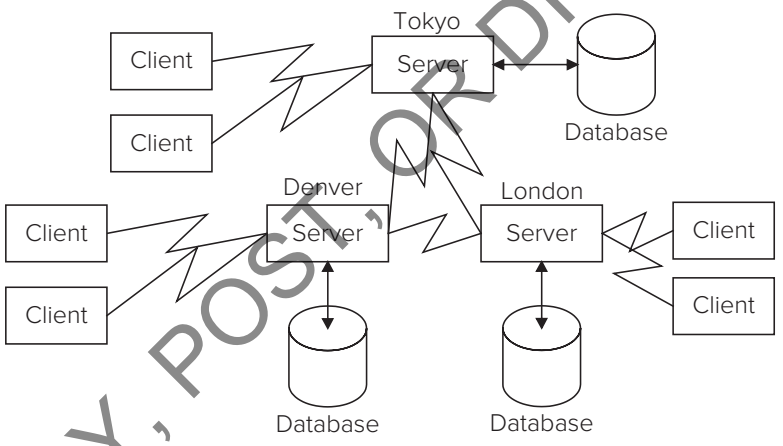


FIGURE 1.15
Distributed Database with Three Sites



The architectures presented in this section assume a traditional product licensing and hosting approach. Cloud computing provides a new approach without initial product licensing costs and no hosting requirements. Using web-based interfaces, organizations can design and deploy databases with dynamic resource allocation provided by the cloud as depicted in Figure 1.16. The cloud service may restrict the design flexibility for database design and operations available for database usage. Internally, the cloud can use any distributed processing approach although the internal details of the cloud are invisible to organizations using the cloud service.

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 subsection describes possible interactions that you may have with a database in an organization. The second subsection describes approaches to plan and control 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 data resources. Chapter 16 provides more detail about the tools and processes used in these management roles.

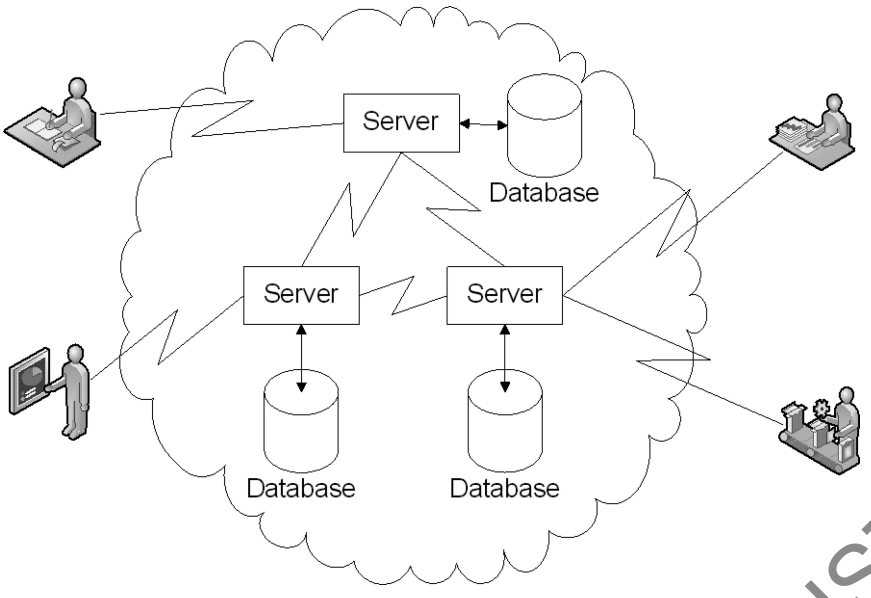


FIGURE 1.16
Cloud-Based Database Architecture

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 1.17 distinguishes between functional users who interact with databases as part of their work and information systems professionals who 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 implementing 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 database usage 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. Analysts/programmers are responsible for collecting

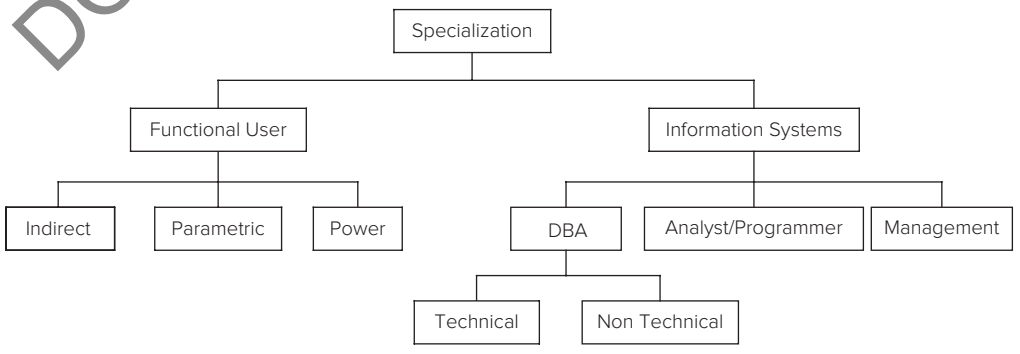


FIGURE 1.17
Classification of Roles

TABLE 1-4
Responsibilities of the Database Administrator

Technical	Non-technical
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
Managing security for database usage	Planning new databases
Troubleshooting database problems	

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. Information systems professionals in analyst/programmer roles should know database development and application development in Parts 3 to 5 of this book.

Database Administrator
a support position specializing 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 non-technical responsibilities (Table 1-4). Technical skills are more detail-oriented; non-technical responsibilities are more people-oriented. The primary technical responsibility is database design. On the non-technical 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. Chapter 16 provides more details about the responsibilities and tools of database administrators.

1.5.2 Managing Data Resources in Organizations

Organizations have used two approaches to manage data resources. The more established approach, information resource management, focuses on 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 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.

Due to the rapid growth of electronic commerce and financial scandals in the 2000s, data governance has emerged as a complementary approach for managing data resources. According to the Data Governance Institute (www.dgi.com), "data governance is the exercise of decision-making and authority for data-related matters." Data governance provides a system of checks and balances to develop data rules and policies, support the application of data rules and policies, and evaluate compliance with data rules and policies. Organizations use the artifacts of data governance to mitigate risks associated with the complex regulatory environment, information security, and information privacy, especially for personal identifiable data and related business transactions.

Data Administrator
a management position that performs planning and policy setting for 'an organization's data resources.

As part of controlling data resources, new management responsibilities have been created in many organizations. The **data administrator** is a management role with responsibilities to plan the development of new databases and control usage of data throughout an organization. The data administrator maintains an enterprise data architecture that describes existing databases and new databases, evaluates new information technologies and determines standards for managing databases. The data administrator supports data governance through participation in the data governance organization and consultation on activities managed by the data governance office.

The data administrator role typically has broader responsibilities than the database administrator role. A data administrator primarily has planning and policy setting roles, while a database administrator has a more technical role focused on individual databases and DBMSs. A data administrator also views data resources in a broader context and considers all kinds of data, both traditional business data and non-traditional unstructured data such as images, videos, and social media. A major effort in many organizations is to develop a data governance program to manage risks associated with the usage of corporate data assets. Data administrators typically assume a leadership role in the data governance program, while database administrators serve in support roles by implementing controls for data governance policies.

Because of broader responsibilities, the data administrator typically is higher in an organization chart. Figure 1.18 depicts two possible placements of data administrators and database administrators. In a small organization, both roles may be combined in systems administration.

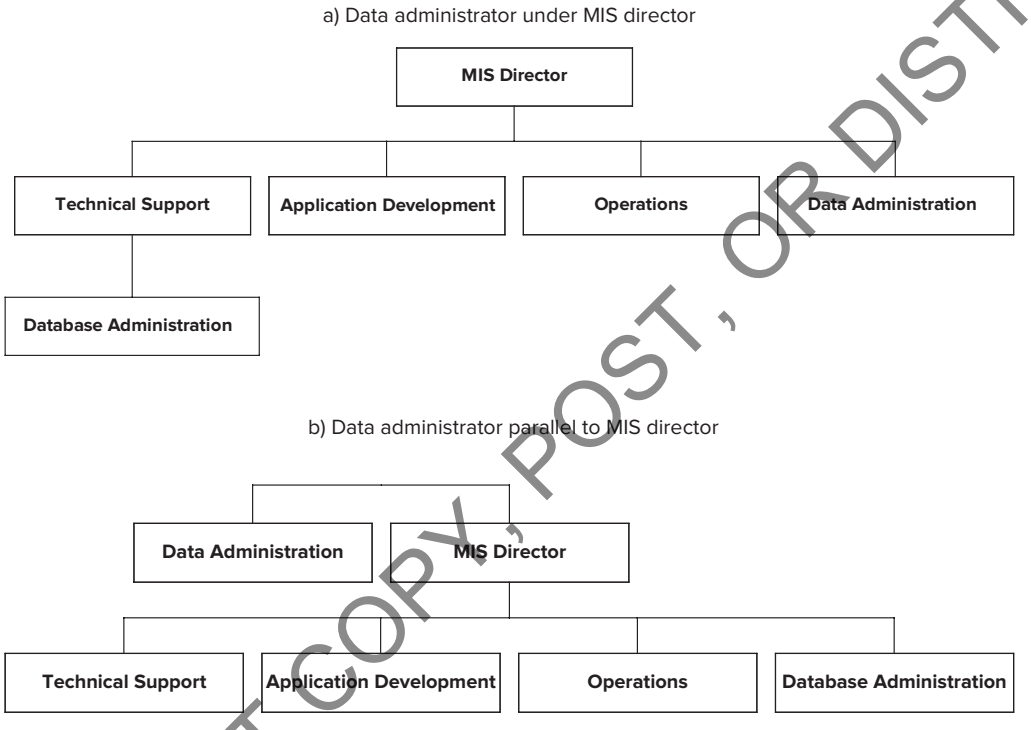


FIGURE 1.18
Organizational Placement of Data and Database Administration

CLOSING THOUGHTS

Chapter 1 has provided a broad introduction to DBMSs. You should use this background as a context for the skills and knowledge 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 the 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, programming language interface, 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, parallel database processing, and distributed databases 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 advanced query formulation skills for database developers. 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
- Structured Query Language (SQL), an industry-standard language for database definition, manipulation, and control
- Transaction: a unit of work that should be processed reliably
- Procedural language interface for combining nonprocedural access with a programming language such as Java 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, data warehouses, and big data
- Types of DBMSs: enterprise, desktop, embedded
- Impact of big data demands and NoSQL database technology to deal with big data challenges
- Data independence to alleviate problems with maintenance of computer programs
- Three Schema Architecture for reducing the impact of database definition changes
- Client-server processing, parallel database processing, and distributed database processing for using databases over computer networks
- Cloud-based database architecture for scalable, on-demand database services without ownership costs and risks
- Database specialist roles: database administrator and data administrator
- Information resource management for utilizing information technology
- Data governance for mitigating risks associated with the complex regulatory environment, information security, and information privacy

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.