part 1

Introduction to Databases

This page intentionally left blank

Databases and Database Users

Databases and database systems are an essential component of life in modern society: most of us encounter several activities every day that involve some interaction with a database. For example, if we go to the bank to deposit or withdraw funds, if we make a hotel or airline reservation, if we access a computerized library catalog to search for a bibliographic item, or if we purchase something online—such as a book, toy, or computer—chances are that our activities will involve someone or some computer program accessing a database. Even purchasing items at a supermarket often automatically updates the database that holds the inventory of grocery items.

These interactions are examples of what we may call **traditional database applications,** in which most of the information that is stored and accessed is either textual or numeric. In the past few years, advances in technology have led to exciting new applications of database systems. New media technology has made it possible to store images, audio clips, and video streams digitally. These types of files are becoming an important component of **multimedia databases. Geographic information systems (GIS)** can store and analyze maps, weather data, and satellite images. **Data warehouses** and **online analytical processing (OLAP)** systems are used in many companies to extract and analyze useful business information from very large databases to support decision making. **Real-time** and **active database technology** is used to control industrial and manufacturing processes. And database search techniques are being applied to the World Wide Web to improve the search for information that is needed by users browsing the Internet.

To understand the fundamentals of database technology, however, we must start from the basics of traditional database applications. In Section 1.1 we start by defining a database, and then we explain other basic terms. In Section 1.2, we provide a

simple UNIVERSITY database example to illustrate our discussion. Section 1.3 describes some of the main characteristics of database systems, and Sections 1.4 and 1.5 categorize the types of personnel whose jobs involve using and interacting with database systems. Sections 1.6, 1.7, and 1.8 offer a more thorough discussion of the various capabilities provided by database systems and discuss some typical database applications. Section 1.9 summarizes the chapter.

The reader who desires a quick introduction to database systems can study Sections 1.1 through 1.5, then skip or browse through Sections 1.6 through 1.8 and go on to Chapter 2.

1.1 Introduction

Databases and database technology have a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science. The word *database* is so commonly used that we must begin by defining what a database is. Our initial definition is quite general.

A **database** is a collection of related data.[1] By **data**, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book or you may have stored it on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database.

The preceding definition of database is quite general; for example, we may consider the collection of words that make up this page of text to be related data and hence to constitute a database. However, the common use of the term *database* is usually more restricted. A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the **miniworld** or the **universe of discourse (UoD)**. Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

In other words, a database has some source from which data is derived, some degree of interaction with events in the real world, and an audience that is actively inter-

---

[1] We will use the word *data* as both singular and plural, as is common in database literature; the context will determine whether it is singular or plural. In standard English, *data* is used for plural and *datum* for singular.

ested in its contents. The end users of a database may perform business transactions (for example, a customer buys a camera) or events may happen (for example, an employee has a baby) that cause the information in the database to change. In order for a database to be accurate and reliable at all times, it must be a true reflection of the miniworld that it represents; therefore, changes must be reflected in the database as soon as possible.

A database can be of any size and complexity. For example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with a simple structure. On the other hand, the computerized catalog of a large library may contain half a million entries organized under different categories—by primary author's last name, by subject, by book title—with each category organized alphabetically. A database of even greater size and complexity is maintained by the Internal Revenue Service (IRS) to monitor tax forms filed by U.S. taxpayers. If we assume that there are 100 million taxpayers and each taxpayer files an average of five forms with approximately 400 characters of information per form, we would have a database of $100 \times 10^6 \times 400 \times 5$ characters (bytes) of information. If the IRS keeps the past three returns of each taxpayer in addition to the current return, we would have a database of 8×10^{11} bytes (800 gigabytes). This huge amount of information must be organized and managed so that users can search for, retrieve, and update the data as needed.

An example of a large commercial database is Amazon.com. It contains data for over 20 million books, CDs, videos, DVDs, games, electronics, apparel, and other items. The database occupies over 2 terabytes (a terabyte is 10^{12} bytes worth of storage) and is stored on 200 different computers (called servers). About 15 million visitors access Amazon.com each day and use the database to make purchases. The database is continually updated as new books and other items are added to the inventory and stock quantities are updated as purchases are transacted. About 100 people are responsible for keeping the Amazon database up-to-date.

A database may be generated and maintained manually or it may be computerized. For example, a library card catalog is a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system. We are only concerned with computerized databases in this book.

A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating,* and *sharing* databases among various users and applications. **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data. Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS. **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the

miniworld, and generating reports from the data. **Sharing** a database allows multiple users and programs to access the database simultaneously.

An **application program** accesses the database by sending queries or requests for data to the DBMS. A **query**[2] typically causes some data to be retrieved; a **transaction** may cause some data to be read and some data to be written into the database.

Other important functions provided by the DBMS include *protecting* the database and *maintaining* it over a long period of time. **Protection** includes *system protection* against hardware or software malfunction (or crashes) and *security protection* against unauthorized or malicious access. A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

It is not absolutely necessary to use general-purpose DBMS software to implement a computerized database. We could write our own set of programs to create and maintain the database, in effect creating our own *special-purpose* DBMS software. In either case—whether we use a general-purpose DBMS or not—we usually have to deploy a considerable amount of complex software. In fact, most DBMSs are very complex software systems.

To complete our initial definitions, we will call the database and DBMS software together a **database system.** Figure 1.1 illustrates some of the concepts we have discussed so far.

1.2 An Example

Let us consider a simple example that most readers may be familiar with: a UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment. Figure 1.2 shows the database structure and a few sample data for such a database. The database is organized as five files, each of which stores **data records** of the same type.[3] The STUDENT file stores data on each student, the COURSE file stores data on each course, the SECTION file stores data on each section of a course, the GRADE_REPORT file stores the grades that students receive in the various sections they have completed, and the PREREQUISITE file stores the prerequisites of each course.

To *define* this database, we must specify the structure of the records of each file by specifying the different types of **data elements** to be stored in each record. In Figure 1.2, each STUDENT record includes data to represent the student's Name, Student_number, Class (such as freshman or '1', sophomore or '2', and so forth), and

---

[2]The term *query*, originally meaning a question or an inquiry, is loosely used for all types of interactions with databases, including modifying the data.

[3]We use the term *file* informally here. At a conceptual level, a *file* is a *collection* of records that may or may not be ordered.

Users/Programmers

Database System

Application Programs/Queries

DBMS Software

Software to Process Queries/Programs

Software to Access Stored Data

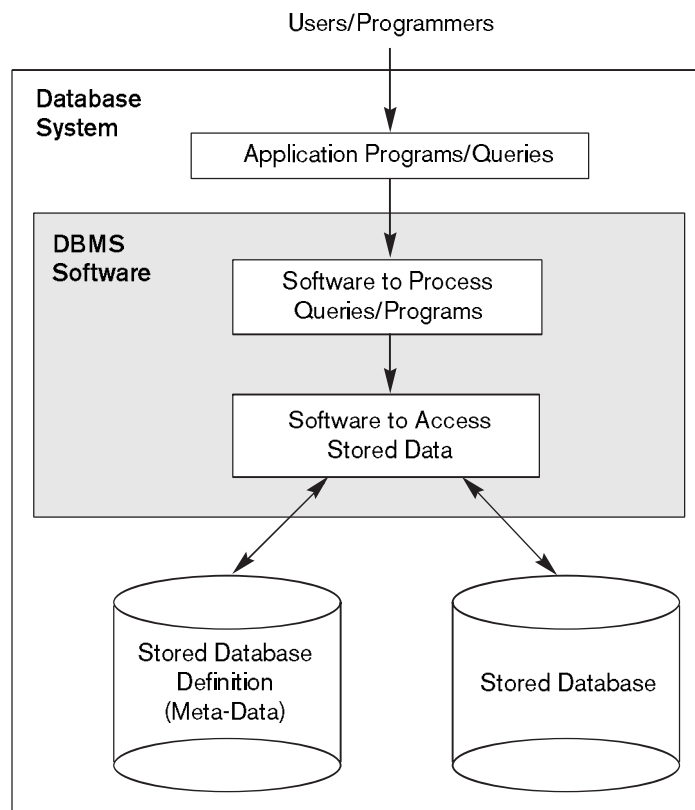Stored Database Definition (Meta-Data)

Stored Database

Figure 1.1
A simplified database system environment.

Major (such as mathematics or 'MATH' and computer science or 'CS'); each COURSE record includes data to represent the Course_name, Course_number, Credit_hours, and Department (the department that offers the course); and so on. We must also specify a **data type** for each data element within a record. For example, we can specify that Name of **STUDENT** is a string of alphabetic characters, Student_number of **STUDENT** is an integer, and Grade of **GRADE_REPORT** is a single character from the set {'A', 'B', 'C', 'D', 'F', 'I'}. We may also use a coding scheme to represent the values of a data item. For example, in Figure 1.2 we represent the Class of a **STUDENT** as 1 for freshman, 2 for sophomore, 3 for junior, 4 for senior, and 5 for graduate student.

To *construct* the **UNIVERSITY** database, we store data to represent each student, course, section, grade report, and prerequisite as a record in the appropriate file. Notice that records in the various files may be related. For example, the record for Smith in the **STUDENT** file is related to two records in the **GRADE_REPORT** file that specify Smith's grades in two sections. Similarly, each record in the **PREREQUISITE** file relates two course records: one representing the course and the other representing the prerequisite. Most medium-size and large databases include many types of records and have *many relationships* among the records.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Database *manipulation* involves querying and updating. Examples of queries are as follows:

- Retrieve the transcript—a list of all courses and grades—of 'Smith'
- List the names of students who took the section of the 'Database' course offered in fall 2008 and their grades in that section
- List the prerequisites of the 'Database' course

Examples of updates include the following:

- Change the class of 'Smith' to sophomore
- Create a new section for the 'Database' course for this semester
- Enter a grade of 'A' for 'Smith' in the 'Database' section of last semester

These informal queries and updates must be specified precisely in the query language of the DBMS before they can be processed.

At this stage, it is useful to describe the database as a part of a larger undertaking known as an information system within any organization. The Information Technology (IT) department within a company designs and maintains an information system consisting of various computers, storage systems, application software, and databases. Design of a new application for an existing database or design of a brand new database starts off with a phase called **requirements specification and analysis**. These requirements are documented in detail and transformed into a **conceptual design** that can be represented and manipulated using some computerized tools so that it can be easily maintained, modified, and transformed into a database implementation. (We will introduce a model called the Entity-Relationship model in Chapter 7 that is used for this purpose.) The design is then translated to a **logical design** that can be expressed in a data model implemented in a commercial DBMS. (In this book we will emphasize a data model known as the Relational Data Model from Chapter 3 onward. This is currently the most popular approach for designing and implementing databases using relational DBMSs.) The final stage is **physical design**, during which further specifications are provided for storing and accessing the database. The database design is implemented, populated with actual data, and continuously maintained to reflect the state of the miniworld.

1.3 Characteristics of the Database Approach

A number of characteristics distinguish the database approach from the much older approach of programming with files. In traditional **file processing**, each user defines and implements the files needed for a specific software application as part of programming the application. For example, one user, the *grade reporting office,* may keep files on students and their grades. Programs to print a student's transcript and to enter new grades are implemented as part of the application. A second user, the *accounting office,* may keep track of students' fees and their payments. Although both users are interested in data about students, each user maintains separate files— and programs to manipulate these files—because each requires some data not avail-

able from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.

In the database approach, a single repository maintains data that is defined once and then accessed by various users. In file systems, each application is free to name data elements independently. In contrast, in a database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications. The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

We describe each of these characteristics in a separate section. We will discuss additional characteristics of database systems in Sections 1.6 through 1.8.

1.3.1 Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data,** and it describes the structure of the primary database (Figure 1.1).

The catalog is used by the DBMS software and also by database users who need information about the database structure. A general-purpose DBMS software package is not written for a specific database application. Therefore, it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with *any number of database applications*—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.

In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only *one specific database,* whose structure is declared in the application programs. For example, an application program written in C++ may have struct or class declarations, and a COBOL program has data division statements to define its files. Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the database definitions from the catalog and using these definitions.

For the example shown in Figure 1.2, the DBMS catalog will store the definitions of all the files shown. Figure 1.3 shows some sample entries in a database catalog.

These definitions are specified by the database designer prior to creating the actual database and are stored in the catalog. Whenever a request is made to access, say, the Name of a STUDENT record, the DBMS software refers to the catalog to determine the structure of the STUDENT file and the position and size of the Name data item within a STUDENT record. By contrast, in a typical file-processing application, the file structure and, in the extreme case, the exact location of Name within a STUDENT record are already coded within each program that accesses this data item.

1.3.2 Insulation between Programs and Data, and Data Abstraction

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require *changing all programs* that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property **program-data independence**.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
.....	.....	......
.....	.....	......
.....	.....	......
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors.
XXXXNNNN is used to define a type with four alpha characters followed by four digits.

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

For example, a file access program may be written in such a way that it can access only STUDENT records of the structure shown in Figure 1.4. If we want to add another piece of data to each STUDENT record, say the Birth_date, such a program will no longer work and must be changed. By contrast, in a DBMS environment, we only need to change the description of STUDENT records in the catalog (Figure 1.3) to reflect the inclusion of the new data item Birth_date; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used.

In some types of database systems, such as object-oriented and object-relational systems (see Chapter 11), users can define operations on data as part of the database definitions. An **operation** (also called a *function* or *method*) is specified in two parts. The *interface* (or *signature*) of an operation includes the operation name and the data types of its arguments (or parameters). The *implementation* (or *method*) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed **program-operation independence.**

The characteristic that allows program-data independence and program-operation independence is called **data abstraction.** A DBMS provides users with a **conceptual representation** of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a **data model** is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model *hides* storage and implementation details that are not of interest to most database users.

For example, reconsider Figures 1.2 and 1.3. The internal implementation of a file may be defined by its record length—the number of characters (bytes) in each record—and each data item may be specified by its starting byte within a record and its length in bytes. The STUDENT record would thus be represented as shown in Figure 1.4. But a typical database user is not concerned with the location of each data item within a record or its length; rather, the user is concerned that when a reference is made to Name of STUDENT, the correct value is returned. A conceptual representation of the STUDENT records is shown in Figure 1.2. Many other details of file storage organization—such as the access paths specified on a file—can be hidden from database users by the DBMS; we discuss storage details in Chapters 17 and 18.

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
Student_number	31	4
Class	35	1
Major	36	4

Figure 1.4
Internal storage format for a STUDENT record, based on the database catalog in Figure 1.3.

In the database approach, the detailed structure and organization of each file are stored in the catalog. Database users and application programs refer to the conceptual representation of the files, and the DBMS extracts the details of file storage from the catalog when these are needed by the DBMS file access modules. Many data models can be used to provide this data abstraction to database users. A major part of this book is devoted to presenting various data models and the concepts they use to abstract the representation of data.

In object-oriented and object-relational databases, the abstraction process includes not only the data structure but also the operations on the data. These operations provide an abstraction of miniworld activities commonly understood by the users. For example, an operation CALCULATE_GPA can be applied to a STUDENT object to calculate the grade point average. Such operations can be invoked by the user queries or application programs without having to know the details of how the operations are implemented. In that sense, an abstraction of the miniworld activity is made available to the user as an **abstract operation.**

1.3.3 Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or **view** of the database. A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example, one user of the database of Figure 1.2 may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure 1.5(a). A second user, who is interested only in checking that students have taken all the prerequisites of each course for which they register, may require the view shown in Figure 1.5(b).

1.3.4 Sharing of Data and Multiuser Transaction Processing

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called **online transaction processing (OLTP)** applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

The concept of a **transaction** has become central to many database applications. A transaction is an *executing program* or *process* that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions. The DBMS must enforce several transaction

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

(a)

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

(b)

Figure 1.5
Two views derived from the database in Figure 1.2. (a) The TRANSCRIPT view.
(b) The COURSE_PREREQUISITES view.

properties. The **isolation** property ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently. The **atomicity** property ensures that either all the database operations in a transaction are executed or none are. We discuss transactions in detail in Part 9.

The preceding characteristics are important in distinguishing a DBMS from traditional file-processing software. In Section 1.6 we discuss additional features that characterize a DBMS. First, however, we categorize the different types of people who work in a database system environment.

1.4 Actors on the Scene

For a small personal database, such as the list of addresses discussed in Section 1.1, one person typically defines, constructs, and manipulates the database, and there is no sharing. However, in large organizations, many people are involved in the design, use, and maintenance of a large database with hundreds of users. In this section we identify the people whose jobs involve the day-to-day use of a large database; we call them the *actors on the scene*. In Section 1.5 we consider people who may be called *workers behind the scene*—those who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job.

1.4.1 Database Administrators

In any organization where many people use the same resources, there is a need for a chief administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the **database administrator (DBA)**. The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.

1.4.2 Database Designers

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements. In many cases, the designers are on the staff of the DBA and may be assigned other staff responsibilities after the database design is completed. Database designers typically interact with each potential group of users and develop **views** of the database that meet the data and processing requirements of these groups. Each view is then analyzed and *integrated* with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

1.4.3 End Users

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

- **Casual end users** occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.
- **Naive** or **parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates—called **canned transactions**—that have been carefully programmed and tested. The tasks that such users perform are varied:
  - Bank tellers check account balances and post withdrawals and deposits.
  - Reservation agents for airlines, hotels, and car rental companies check availability for a given request and make reservations.

      □ Employees at receiving stations for shipping companies enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.

■ **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

■ **Standalone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes.

A typical DBMS provides multiple facilities to access a database. Naive end users need to learn very little about the facilities provided by the DBMS; they simply have to understand the user interfaces of the standard transactions designed and implemented for their use. Casual users learn only a few facilities that they may use repeatedly. Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements. Standalone users typically become very proficient in using a specific software package.

1.4.4 System Analysts and Application Programmers (Software Engineers)

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements. **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers—commonly referred to as **software developers** or **software engineers**—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

1.5 Workers behind the Scene

In addition to those who design, use, and administer a database, others are associated with the design, development, and operation of the DBMS *software and system environment.* These persons are typically not interested in the database content itself. We call them the *workers behind the scene,* and they include the following categories:

■ **DBMS system designers and implementers** design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or **modules,** including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security. The DBMS must interface with other system software such as the operating system and compilers for various programming languages.

- **Tool developers** design and implement **tools**—the software packages that facilitate database modeling and design, database system design, and improved performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation. In many cases, independent software vendors develop and market these tools.
- **Operators and maintenance personnel** (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Although these categories of workers behind the scene are instrumental in making the database system available to end users, they typically do not use the database contents for their own purposes.

1.6 Advantages of Using the DBMS Approach

In this section we discuss some of the advantages of using a DBMS and the capabilities that a good DBMS should possess. These capabilities are in addition to the four main characteristics discussed in Section 1.3. The DBA must utilize these capabilities to accomplish a variety of objectives related to the design, administration, and use of a large multiuser database.

1.6.1 Controlling Redundancy

In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications. For example, consider the UNIVERSITY database example of Section 1.2; here, two groups of users might be the course registration personnel and the accounting office. In the traditional approach, each group independently keeps files on students. The accounting office keeps data on registration and related billing information, whereas the registration office keeps track of student courses and grades. Other groups may further duplicate some or all of the same data in their own files.

This **redundancy** in storing the same data multiple times leads to several problems. First, there is the need to perform a single logical update—such as entering data on a new student—multiple times: once for each file where student data is recorded. This leads to *duplication of effort*. Second, *storage space is wasted* when the same data is stored repeatedly, and this problem may be serious for large databases. Third, files that represent the same data may become *inconsistent*. This may happen because an update is applied to some of the files but not to others. Even if an update—such as adding a new student—is applied to all the appropriate files, the data concerning the student may still be *inconsistent* because the updates are applied independently by each user group. For example, one user group may enter a student's birth date erroneously as 'JAN-19-1988', whereas the other user groups may enter the correct value of 'JAN-29-1988'.

In the database approach, the views of different user groups are integrated during database design. Ideally, we should have a database design that stores each logical data item—such as a student's name or birth date—in *only one place* in the database. This is known as **data normalization**, and it ensures consistency and saves storage space (data normalization is described in Part 6 of the book). However, in practice, it is sometimes necessary to use **controlled redundancy** to improve the performance of queries. For example, we may store Student_name and Course_number redundantly in a GRADE_REPORT file (Figure 1.6(a)) because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier. By placing all the data together, we do not have to search multiple files to collect this data. This is known as **denormalization**. In such cases, the DBMS should have the capability to *control* this redundancy in order to prohibit inconsistencies among the files. This may be done by automatically checking that the Student_name–Student_number values in any GRADE_REPORT record in Figure 1.6(a) match one of the Name–Student_number values of a STUDENT record (Figure 1.2). Similarly, the Section_identifier–Course_number values in GRADE_REPORT can be checked against SECTION records. Such checks can be specified to the DBMS during database design and automatically enforced by the DBMS whenever the GRADE_REPORT file is updated. Figure 1.6(b) shows a GRADE_REPORT record that is inconsistent with the STUDENT file in Figure 1.2; this kind of error may be entered if the redundancy is *not controlled.* Can you tell which part is inconsistent?

1.6.2 Restricting Unauthorized Access

When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data is often considered confidential, and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas

Figure 1.6

Redundant storage of Student_name and Course_name in GRADE_REPORT. (a) Consistent data. (b) Inconsistent record.

GRADE_REPORT

Student_number	Student_name	Section_identifier	Course_number	Grade
17	Smith	112	MATH2410	B
17	Smith	119	CS1310	C
8	Brown	85	MATH2410	A
8	Brown	92	CS1310	A
8	Brown	102	CS3320	B
8	Brown	135	CS3380	A

(a)

GRADE_REPORT

Student_number	Student_name	Section_identifier	Course_number	Grade
17	Brown	112	MATH2410	B

(b)

others are allowed to retrieve and update. Hence, the type of access operation—retrieval or update—must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. A DBMS should provide a **security and authorization subsystem,** which the DBA uses to create accounts and to specify account restrictions. Then, the DBMS should enforce these restrictions automatically. Notice that we can apply similar controls to the DBMS software. For example, only the dba's staff may be allowed to use certain **privileged software,** such as the software for creating new accounts. Similarly, parametric users may be allowed to access the database only through the predefined canned transactions developed for their use.

1.6.3 Providing Persistent Storage for Program Objects

Databases can be used to provide **persistent storage** for program objects and data structures. This is one of the main reasons for **object-oriented database systems.** Programming languages typically have complex data structures, such as record types in Pascal or class definitions in C++ or Java. The values of program variables or objects are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage. When the need arises to read this data once more, the programmer must convert from the file format to the program variable or object structure. Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions. Hence, a complex object in C++ can be stored permanently in an object-oriented DBMS. Such an object is said to be **persistent,** since it survives the termination of program execution and can later be directly retrieved by another C++ program.

The persistent storage of program objects and data structures is an important function of database systems. Traditional database systems often suffered from the so-called **impedance mismatch problem,** since the data structures provided by the DBMS were incompatible with the programming language's data structures. Object-oriented database systems typically offer data structure **compatibility** with one or more object-oriented programming languages.

1.6.4 Providing Storage Structures and Search Techniques for Efficient Query Processing

Database systems must provide capabilities for *efficiently executing queries and updates.* Because the database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records. Auxiliary files called **indexes** are used for this purpose. Indexes are typically based on tree data structures or hash data structures that are suitably modified for disk search. In order to process the database records needed by a particular query, those records must be copied from disk to main memory. Therefore, the DBMS often has a **buffering** or **caching** module that maintains parts of the database in main memory buffers. In general, the operating system is responsible for

disk-to-memory buffering. However, because data buffering is crucial to the DBMS performance, most DBMSs do their own data buffering.

The **query processing and optimization** module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures. The choice of which indexes to create and maintain is part of *physical database design and tuning*, which is one of the responsibilities of the DBA staff. We discuss the query processing, optimization, and tuning in Part 8 of the book.

1.6.5 Providing Backup and Recovery

A DBMS must provide facilities for recovering from hardware or software failures. The **backup and recovery subsystem** of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Alternatively, the recovery subsystem could ensure that the transaction is resumed from the point at which it was interrupted so that its full effect is recorded in the database. Disk backup is also necessary in case of a catastrophic disk failure. We discuss recovery and backup in Chapter 23.

1.6.6 Providing Multiple User Interfaces

Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users. Both forms-style interfaces and menu-driven interfaces are commonly known as **graphical user interfaces (GUIs)**. Many specialized languages and environments exist for specifying GUIs. Capabilities for providing Web GUI interfaces to a database—or Web-enabling a database—are also quite common.

1.6.7 Representing Complex Relationships among Data

A database may include numerous varieties of data that are interrelated in many ways. Consider the example shown in Figure 1.2. The record for 'Brown' in the STUDENT file is related to four records in the GRADE_REPORT file. Similarly, each section record is related to one course record and to a number of GRADE_REPORT records—one for each student who completed that section. A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

1.6.8 Enforcing Integrity Constraints

Most database applications have certain **integrity constraints** that must hold for the data. A DBMS should provide capabilities for defining and enforcing these con-

straints. The simplest type of integrity constraint involves specifying a data type for each data item. For example, in Figure 1.3, we specified that the value of the Class data item within each STUDENT record must be a one digit integer and that the value of Name must be a string of no more than 30 alphabetic characters. To restrict the value of Class between 1 and 5 would be an additional constraint that is not shown in the current catalog. A more complex type of constraint that frequently occurs involves specifying that a record in one file must be related to records in other files. For example, in Figure 1.2, we can specify that *every section record must be related to a course record*. This is known as a **referential integrity** constraint. Another type of constraint specifies uniqueness on data item values, such as *every course record must have a unique value for Course_number. This is known as a* **key** or **uniqueness** constraint. These constraints are derived from the meaning or **semantics** of the data and of the miniworld it represents. It is the responsibility of the database designers to identify integrity constraints during database design. Some constraints can be specified to the DBMS and automatically enforced. Other constraints may have to be checked by update programs or at the time of data entry. For typical large applications, it is customary to call such constraints **business rules.**

A data item may be entered erroneously and still satisfy the specified integrity constraints. For example, if a student receives a grade of 'A' but a grade of 'C' is entered in the database, the DBMS *cannot* discover this error automatically because 'C' is a valid value for the Grade data type. Such data entry errors can only be discovered manually (when the student receives the grade and complains) and corrected later by updating the database. However, a grade of 'Z' would be rejected automatically by the DBMS because 'Z' is not a valid value for the Grade data type. When we discuss each data model in subsequent chapters, we will introduce rules that pertain to that model implicitly. For example, in the Entity-Relationship model in Chapter 7, a relationship must involve at least two entities. Such rules are **inherent rules** of the data model and are automatically assumed to guarantee the validity of the model.

1.6.9 Permitting Inferencing and Actions Using Rules

Some database systems provide capabilities for defining *deduction rules* for *inferencing* new information from the stored database facts. Such systems are called **deductive database systems**. For example, there may be complex rules in the miniworld application for determining when a student is on probation. These can be specified *declaratively* as **rules**, which when compiled and maintained by the DBMS can determine all students on probation. In a traditional DBMS, an explicit *procedural program code* would have to be written to support such applications. But if the miniworld rules change, it is generally more convenient to change the declared deduction rules than to recode procedural programs. In today's relational database systems, it is possible to associate **triggers** with tables. A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on. More involved procedures to enforce rules are popularly called **stored procedures**; they become a part of the overall database definition and are invoked appropriately when certain conditions are met. More powerful functionality is provided by **active database systems**, which

provide active rules that can automatically initiate actions when certain events and conditions occur.

1.6.10 Additional Implications of Using the Database Approach

This section discusses some additional implications of using the database approach that can benefit most organizations.

Potential for Enforcing Standards. The database approach permits the DBA to define and enforce standards among database users in a large organization. This facilitates communication and cooperation among various departments, projects, and users within the organization. Standards can be defined for names and formats of data elements, display formats, report structures, terminology, and so on. The DBA can enforce standards in a centralized database environment more easily than in an environment where each user group has control of its own data files and software.

Reduced Application Development Time. A prime selling feature of the database approach is that developing a new application—such as the retrieval of certain data from the database for printing a new report—takes very little time. Designing and implementing a large multiuser database from scratch may take more time than writing a single specialized file application. However, once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities. Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system.

Flexibility. It may be necessary to change the structure of a database as requirements change. For example, a new user group may emerge that needs information not currently in the database. In response, it may be necessary to add a file to the database or to extend the data elements in an existing file. Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs.

Availability of Up-to-Date Information. A DBMS makes the database available to all users. As soon as one user's update is applied to the database, all other users can immediately see this update. This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases, and it is made possible by the concurrency control and recovery subsystems of a DBMS.

Economies of Scale. The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments as well as redundancies among applications. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its own (lower performance) equipment. This reduces overall costs of operation and management.