CHAPTER 9

Database Systems

A database is a system that converts a large collection of data into an abstract tool, allowing users to search for and extract pertinent items of information in a manner that is convenient to the user. In this chapter we explore this subject as well as take side excursions into the related fields of data mining, which seeks techniques for uncovering hidden patterns in large data collections, and traditional file structures, which provide many of the tools underlying today's database and data mining systems.

9.1 Database Fundamentals
The Significance of Database Systems
The Role of Schemas
Database Management Systems
Database Models

9.2 The Relational Model
Issues of Relational Design
Relational Operations
SQL

***9.3 Object-Oriented Databases**

***9.4 Maintaining Database Integrity**
The Commit/Rollback Protocol
Locking

***9.5 Traditional File Structures**
Sequential Files
Indexed Files
Hash Files

9.6 Data Mining

9.7 Social Impact of Database Technology

**Asterisks indicate suggestions for optional sections.*

Today's technology is capable of storing extremely large amounts of data, but such data collections are useless unless we are able to extract those particular items of information that are pertinent to the task at hand. In this chapter we will study database systems and learn how these systems apply abstraction to convert large data conglomerates into useful information sources. As a related topic, we will investigate the rapidly expanding field of data mining, whose goal is to develop techniques for identifying and exploring patterns within data collections. We will also examine the principles of traditional file structures, which provide the underpinnings for today's database and data mining systems.

9.1 Database Fundamentals

The term **database** refers to a collection of data that is multidimensional in the sense that internal links between its entries make the information accessible from a variety of perspectives. This is in contrast to a traditional file system (Section 9.5), sometimes called a **flat file,** which is a one-dimensional storage system, meaning that it presents its information from a single point of view. Whereas a flat file containing information about composers and their compositions might provide a list of compositions arranged by composer, a database might present all the works by a single composer, all the composers who wrote a particular type of music, and perhaps the composers who wrote variations of another composer's work.

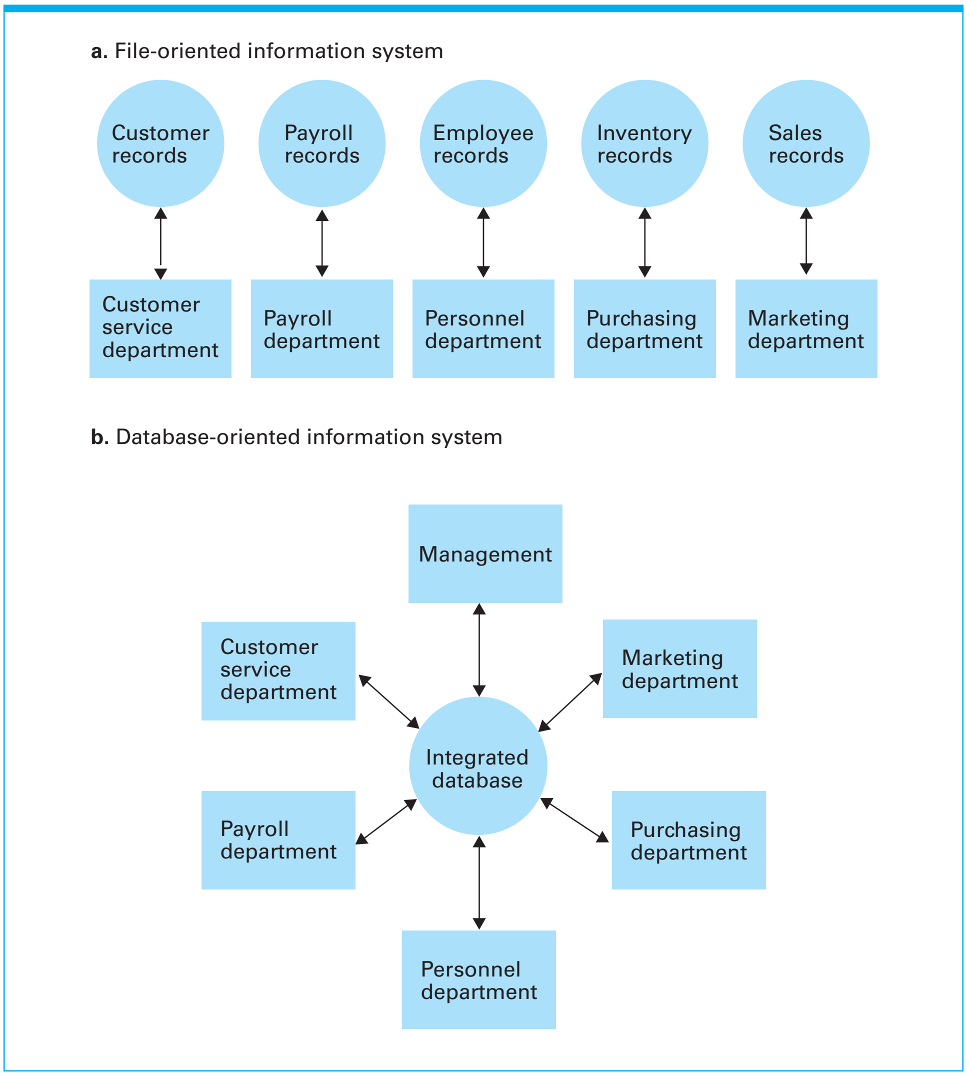
The Significance of Database Systems

Historically, as computing machinery found broader uses in information management, each application tended to be implemented as a separate system with its own collection of data. Payroll was processed using the payroll file, the personnel department maintained its own employee records, and inventory was managed via an inventory file. This meant that much of the information required by an organization was duplicated throughout the company, while many different but related items were stored in separate systems. In this setting, database systems emerged as a means of integrating the information stored and maintained by a particular organization (Figure 9.1). With such a system, the same sales data could be used to produce restocking orders; create reports on market trends, direct advertisements, and product announcements to customers who are most likely to respond favorably to such information; and generate bonus checks for members of the sales force.

Such integrated pools of information provided a valuable resource with which management decisions could be made, assuming the information could be accessed in a meaningful way. In turn, database research focused on developing techniques by which the information in a database could be brought to the decision-making process. Much progress has been made in this regard. Today, database technology, combined with data mining techniques, is an important management tool, allowing the management of an organization to extract pertinent information from enormous amounts of data covering all aspects of the organization and its environment.

Moreover, database systems have become the underlying technology that supports many of the more popular sites on the World Wide Web. The underlying theme of sites such as Google, eBay, and Amazon is to provide an interface

Figure 9.1 A file versus a database organization



between clients and databases. To respond to a client's request, the server interrogates a database, organizes the results in the form of a Web page, and sends that page to the client. Such Web interfaces have popularized a new role for database technology in which a database is no longer a means of storing a company's records but, instead, is the company's product. Indeed, by combining database technology with Web interfaces, the Internet has become a major world-wide information source.

The Role of Schemas

Among the disadvantages of the proliferation of database technology is the potential of sensitive data being accessed by unauthorized personnel. Someone placing an order at a company's website should not have access to the company's financial data; similarly, an employee in a company's benefits department may need access to the company's employee records but should not have access to the corporation's inventory or sales records. Thus the ability to control access to the information in the database is as important as the ability to share it.

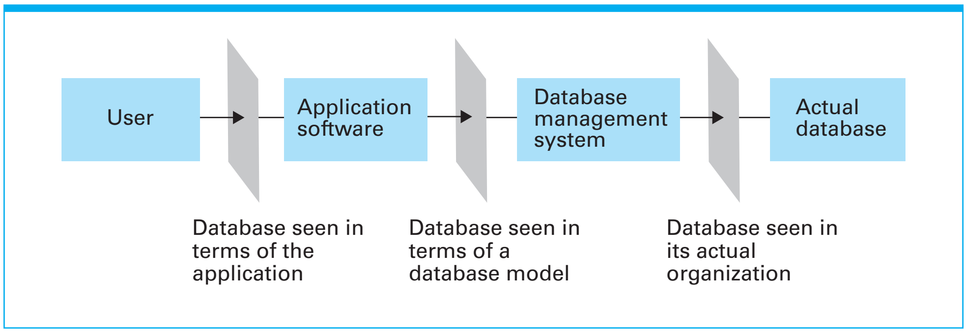
To provide different users access to different information within a database, database systems often rely on schemas and subschemas. A **schema** is a description of the entire database structure that is used by the database software to maintain the database. A **subschema** is a description of only that portion of the database pertinent to a particular user's needs. For example, a schema for a university database would indicate that each student record contains such items as the current address and phone number of that student in addition to the student's academic record. Moreover, it would indicate that each student record is linked to the record of the student's faculty adviser. In turn, the record for each faculty member would contain the person's address, employment history, and so on. Based on this schema, a linkage system would be maintained that ultimately connected the information about a student to the employment history of a faculty member.

To keep the university's registrar from using this linkage to obtain privileged information about the faculty, the registrar's access to the database must be restricted to a subschema whose description of the faculty records does not include employment history. Under this subschema, the registrar could find out which faculty member is a particular student's adviser but could not obtain access to additional information about that faculty member. In contrast, the subschema for the payroll department would provide the employment history of each faculty member but would not include the linkage between students and advisers. Thus the payroll department could modify a faculty member's salary but could not obtain the names of the students advised by that person.

Database Management Systems

A typical database application involves multiple software layers, which we will group into two major layers—an application layer and a database management layer (Figure 9.2). The application software handles the communication with the user of the database and may be quite complex, as exemplified by applications

Figure 9.2 The conceptual layers of a database implementation


User
Application software
Database management system
Actual database
Database seen in terms of the application
Database seen in terms of a database model
Database seen in its actual organization

Distributed Databases

With the advancement of networking capabilities, database systems have grown to encompass databases, known as distributed databases, that consist of data residing on different machines. For instance, an international corporation might store and maintain local employee records at local sites yet link those records via a network to create a single distributed database.

A distributed database might contain fragmented and/or replicated data. The first case is exemplified by the previous employee-records example in which different fragments of the database are stored in different locations. In the second case, duplicates of the same database component are stored at different locations. Such replication might occur as a means of reducing information retrieval time. Both cases pose problems not present in more traditional centralized databases—how to disguise the distributed nature of the database so that it functions as a coherent system or how to ensure that replicated portions of a database remain duplicates of each other as updates occur. In turn, the study of distributed databases is a current area of research.

in which users access a database by means of a website. In that case the entire application layer consists of clients throughout the Internet and a server that uses the database to fill the requests from the clients.

Note that the application software does not directly manipulate the database. The actual manipulation of the database is accomplished by the **database management system (DBMS).** Once the application software has determined what action the user is requesting, it uses the DBMS as an abstract tool to obtain the results. If the request is to add or delete data, it is the DBMS that actually alters the database. If the request is to retrieve information, it is the DBMS that performs the required searches.

This dichotomy between the application software and the DBMS has several benefits. One is that it allows for the construction and use of abstract tools, which we have repeatedly found to be a major simplifying concept in software design. If the details of how the database is actually stored are isolated within the DBMS, the design of the application software can be greatly simplified. For instance, with a well-designed DBMS, the application software does not have to be concerned with whether the database is stored on a single machine or scattered among many machines within a network as a **distributed database.** Instead, the DBMS would deal with these issues, allowing the application software to access the database without concern for where the data is actually stored.

A second advantage of separating the application software from the DBMS is that such an organization provides a means for controlling access to the database. By dictating that the DBMS performs all access to the database, the DBMS can enforce the restrictions imposed by the various subschemas. In particular, the DBMS can use the entire database schema for its internal needs but can require that the application software employed by each user remain within the bounds described by that user's subschema.

Still another reason for separating the user interface and actual data manipulation into two different software layers is to achieve **data independence**—the ability to change the organization of the database itself without changing the application software. For example, the personnel department might need to add an additional field to each employee's record to indicate whether the corresponding employee chose to participate in the company's new health insurance program. If the application software dealt directly with the database, such a change in the data's format could require modifications to all application programs dealing with the database. As a result, the change instigated by the personnel department might cause changes to the payroll program as well as to the program for printing mailing labels for the company's newsletter.

The separation between application software and a DBMS removes the need for such reprogramming. To implement a change in the database required by a single user, one needs to change only the overall schema and the subschemas of those users involved in the change. The subschemas of all the other users remain the same, so their application software, which is based on the unaltered subschemas, does not need to be modified.

Database Models

We have repeatedly seen how abstraction can be used to hide internal complexities. Database management systems provide yet another example. They hide the complexities of a database's internal structure, allowing the user of the database to imagine that the information stored in the database is arranged in a more useful format. In particular, a DBMS contains routines that translate commands stated in terms of a conceptual view of the database into the actions required by the actual data storage system. This conceptual view of the database is called a **database model.**

In the following sections we will consider both the relational database model and the object-oriented database model. In the case of the relational database model, the conceptual view of the database is that of a collection of tables consisting of rows and columns. For example, information about a company's employees might be viewed as a table containing a row for each employee and columns labeled name, address, employee identification number, and so on. In turn, the DBMS would contain routines that would allow the application software to select certain entries from a particular row of the table or perhaps to report the range of values found in the salary column—even though the information is not actually stored in rows and columns.

These routines form the abstract tools used by the application software to access the database. More precisely, application software is often written in one of the general-purpose programming languages, such as those discussed in Chapter 6. These languages provide the basic ingredients for algorithmic expressions but lack instructions for manipulating a database. However, a program written in one of these languages can use the routines provided by the DBMS as prewritten subroutines—in effect extending the capabilities of the language in a manner that supports the conceptual image of the database model.

The search for better database models is an ongoing process. The goal is to find models that allow complex data systems to be conceptualized easily, lead to concise ways of expressing requests for information, and produce efficient database management systems.

Questions & Exercises

1. Identify two departments in a manufacturing plant that would have different uses for the same or similar inventory information. Then, describe how the subschema for the two departments might differ.
2. What is the purpose of a database model?
3. Summarize the roles of the application software and a DBMS.

9.2 The Relational Model

In this section we look more closely at the relational database model. It portrays data as being stored in rectangular tables, called **relations,** which are similar to the format in which information is displayed by spreadsheet programs. For example, the relational model allows information regarding the employees of a firm to be represented by a relation such as that shown in Figure 9.3.

A row in a relation is called a **tuple** (some say "TOO-pul," others say "TUH-pul"). In the relation of Figure 9.3, tuples consist of the information about a particular employee. Columns in a relation are referred to as **attributes** because each entry in a column describes some characteristic, or attribute, of the entity represented by the corresponding tuple.

Issues of Relational Design

A pivotal step in designing a relational database is to design the relations making up the database. Although this might appear to be a simple task, many subtleties are waiting to trap the unwary designer.

Suppose that in addition to the information contained in the relation of Figure 9.3, we want to include information about the jobs held by the employees. We might want to include a job history associated with each employee that consists of such attributes as job title (secretary, office manager, floor supervisor), a job identification code (unique to each job), the skill code associated with each job, the department in which the job exists, and the period during which the employee held the job in terms of a starting date and termination date. (We use an asterisk as the termination date if the job represents the employee's current position.)

Figure 9.3 A relation containing employee information

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555
•	•	•	•
•	•	•	•
•	•	•	•

One approach to this problem is to extend the relation in Figure 9.3 to include these attributes as additional columns in the table, as shown in Figure 9.4. However, close examination of the result reveals several problems. One is a lack of efficiency due to redundancy. The relation no longer contains one tuple for each employee but rather one tuple for each assignment of an employee to a job. If an employee has advanced in the company through a sequence of several jobs, several tuples in the new relation must contain the same information about the employee (name, address, identification number, and Social Security number). For example, the personal information about Baker and Smith is repeated because they have held more than one job. Moreover, when a particular position has been held by numerous employees, the department associated with that job along with the appropriate skill code must be repeated in each tuple representing an assignment of the job. For example, the description of the floor manager job is duplicated because more than one employee has held this position.

Another, perhaps more serious, problem with our extended relation surfaces when we consider deleting information from the database. Suppose for example that Joe E. Baker is the only employee to hold the job identified as D7. If he were to leave the company and be deleted from the database represented in Figure 9.4, we would lose the information about job D7. After all, the only tuple containing the fact that job D7 requires a skill level of K2 is the tuple relating to Joe Baker.

You might argue that the ability to erase only a portion of a tuple could solve the problem, but this would in turn introduce other complications. For instance, should the information relating to job F5 also be retained in a partial tuple, or does this information reside elsewhere in the relation? Moreover, the temptation to use partial tuples is a strong indication that the design of the database can be improved.

The cause of all these problems is that we have combined more than one concept in a single relation. As proposed, the extended relation in Figure 9.4 contains information dealing directly with employees (name, identification number, address, Social Security number), information about the jobs available in the company (job identification, job title, department, skill code), and information

Figure 9.4 A relation containing redundancy

Empl Id	Name	Address	SSN	Job Id	Job Title	Skill Code	Dept	Start Date	Term Date
25X15	Joe E. Baker	33 Nowhere St.	111223333	F5	Floor manager	FM3	Sales	9-1-2009	9-30-2010
25X15	Joe E. Baker	33 Nowhere St.	111223333	D7	Dept. head	K2	Sales	10-1-2010	*
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999	F5	Floor manager	FM3	Sales	10-1-2009	*
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S25X	Secretary	T5	Personnel	3-1-1999	4-30-2010
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555	S26Z	Secretary	T6	Accounting	5-1-2010	*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Database Systems for PCs

Personal computers are used in a variety of applications, ranging from elementary to sophisticated. In elementary "database" applications, such as storing Christmas card lists or maintaining bowling league records, spreadsheet systems are often used in lieu of database software since the application calls for little more than the ability to store, print, and sort data. There are, however, true database systems available for the PC market, one of which is Microsoft's Access. This is a complete relational database system as described in Section 9.2, as well as chart- and report-generation software. Access provides an excellent example of how the principles presented in this text form the backbone of popular products on the market today.

regarding the relationship between employees and jobs (start date, termination date). On the basis of this observation, we can solve our problems by redesigning the system using three relations—one for each of the preceding categories. We can keep the original relation in Figure 9.3 (which we now call the `EMPLOYEE` relation) and insert the additional information in the form of the two new relations called `JOB` and `ASSIGNMENT`, which produces the database in Figure 9.5.

Figure 9.5 An employee database consisting of three relations

EMPLOYEE relation

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555

JOB relation

Job Id	Job Title	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

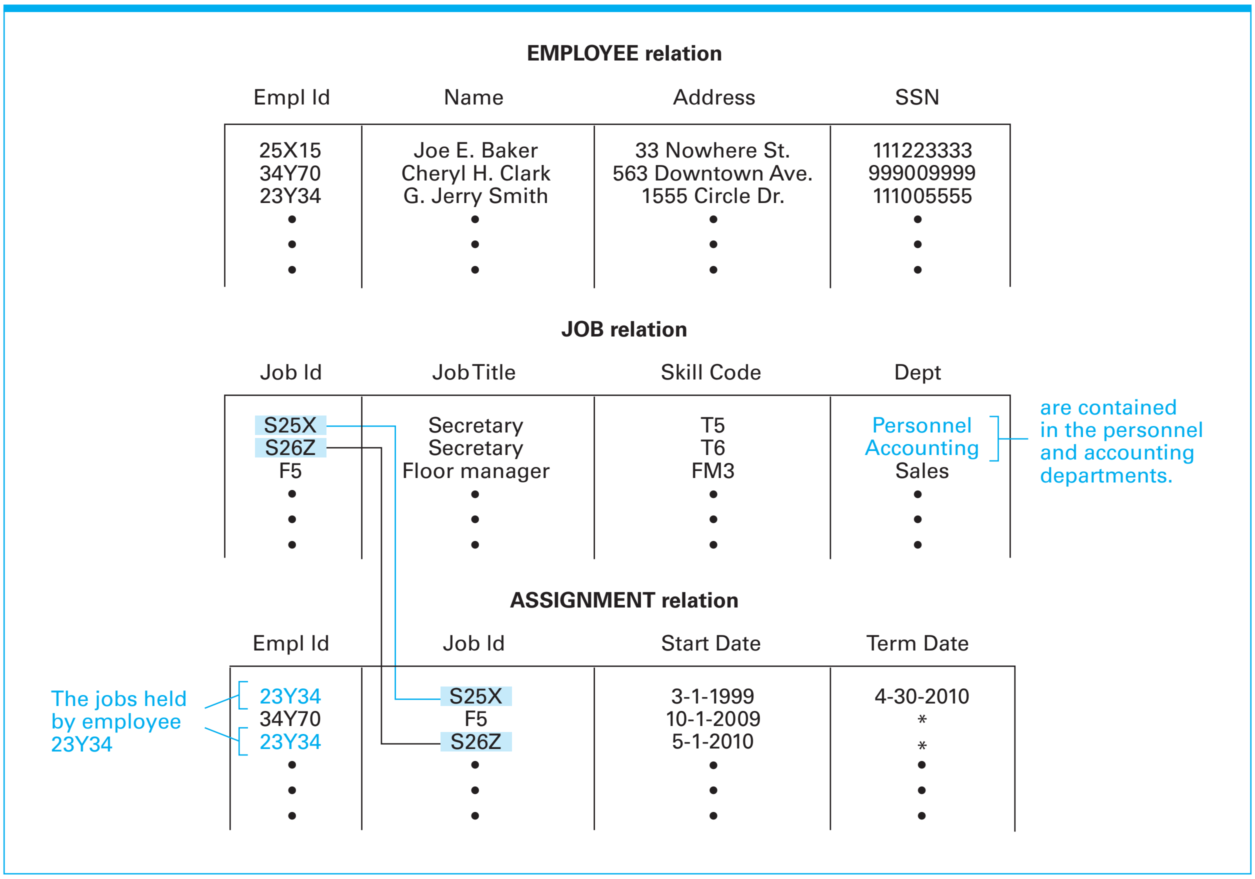
ASSIGNMENT relation

Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
23Y34	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

A database consisting of these three relations contains the pertinent information about employees in the EMPLOYEE relation, about available jobs in the JOB relation, and about job history in the ASSIGNMENT relation. Additional information is implicitly available by combining the information from different relations. For instance, if we know an employee's identification number, we can find the departments in which that employee has worked by first finding all the jobs that employee has held using the ASSIGNMENT relation and then finding the departments associated with those jobs by means of the JOB relation (Figure 9.6). Through processes such as these, any information that could be obtained from the single large relation can be obtained from the three smaller relations without the problems previously cited.

Unfortunately, dividing information into various relations is not always as trouble-free as in the preceding example. For instance, compare the original relation in Figure 9.7, with attributes `EmplId`, `JobTitle`, and `Dept`, to the proposed decomposition into two relations. At first glance, the two-relation system might appear to contain the same information as the single-relation system, but, in fact, it does not. Consider for example the problem of finding the department in which a given employee works. This is easily done in the single-relation system by interrogating the tuple containing the employee identification number of the target employee and extracting the corresponding department. However, in the

Figure 9.6 Finding the departments in which employee 23Y34 has worked



EMPLOYEE relation

Empl Id	Name	Address	SSN
25X15	Joe E. Baker	33 Nowhere St.	111223333
34Y70	Cheryl H. Clark	563 Downtown Ave.	999009999
23Y34	G. Jerry Smith	1555 Circle Dr.	111005555
•	•	•	•
•	•	•	•
•	•	•	•

JOB relation

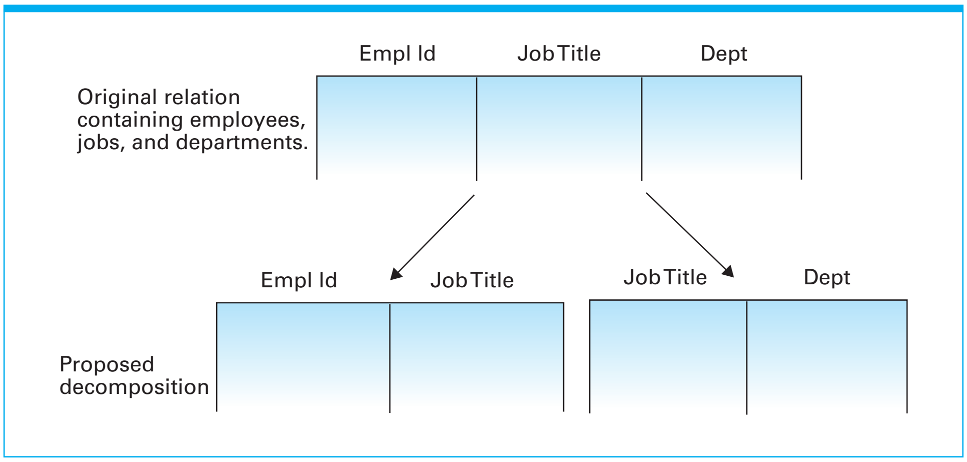
Job Id	JobTitle	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

ASSIGNMENT relation

Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
23Y34	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

are contained in the personnel and accounting departments.
The jobs held by employee 23Y34

Figure 9.7 A relation and a proposed decomposition



two-relation system, the desired information is not necessarily available. We can find the job title of the target employee and a department having such a job but this does not necessarily mean that the target employee works in that particular department, because several departments might have jobs with the same title.

We see, then, that sometimes dividing a relation into smaller relations causes the loss of information, and sometimes it does not. (The latter is called a **lossless decomposition**—or sometimes a **nonloss decomposition.**) Such relational characteristics are important design considerations. The goal is to identify the relational characteristics that can lead to problems in database design and find ways to reorganize those relations to remove these problematic characteristics.

Relational Operations

Now that you have a basic understanding of how data can be organized in terms of the relational model, it is time to see how information can be extracted from a database consisting of relations. We begin with a look at some operations that we might want to perform on relations.

At times we need to select certain tuples from a relation. To retrieve the information about an employee, we must select the tuple with the appropriate identification attribute value from the EMPLOYEE relation, or to obtain a list of the job titles in a certain department, we must select the tuples from the JOB relation having that department as their department attribute. The result of such a process is another relation consisting of the tuples selected from the parent relation. The outcome of selecting information about a particular employee results in a relation containing only one tuple from the EMPLOYEE relation. The outcome of selecting the tuples associated with a certain department results in a relation that probably contains several tuples from the JOB relation.

In short, one operation we might want to perform on a relation is to select tuples possessing certain characteristics and to place these selected tuples in a new relation. To express this operation, we adopt the syntax

```
NEW ← SELECT from EMPLOYEE where EmplId = '34Y70'
```

The semantics of this statement is to create a new relation called NEW containing those tuples (there should be only one in this case) from the relation EMPLOYEE whose EmplId attribute equals 34Y70 (Figure 9.8).

In contrast to the SELECT operation, which extracts rows from a relation, the PROJECT operation extracts columns. Suppose, for example, that in searching for the job titles in a certain department, we had already SELECTed the tuples from the JOB relation that pertained to the target department and placed these tuples in a new relation called NEW1. The list we are seeking is the JobTitle column within this new relation. The PROJECT operation allows us to extract this column (or columns if required) and place the result in a new relation. We express such an operation as

```
NEW2 ← PROJECT JobTitle from NEW1
```

The result is the creation of another new relation (called NEW2) that contains the single column of values from the JobTitle column of relation NEW1.

As another example of the PROJECT operation, the statement

```
MAIL ← PROJECT Name, Address from EMPLOYEE
```

can be used to obtain a listing of the names and addresses of all employees. This list is in the newly created (two-column) relation called MAIL (Figure 9.9).

Another operation used in conjunction with relational databases is the JOIN operation. It is used to combine different relations into one relation. The JOIN of two relations produces a new relation whose attributes consist of the attributes from the original relations (Figure 9.10). The names of these attributes are the same as those in the original relations except that each is prefixed by the relation of its origin. (If relation A containing attributes V and W is JOINed with relation B containing attributes X, Y, and Z, then the result has five attributes named A.V, A.W, B.X, B.Y, and B.Z.) This naming convention ensures that the attributes in the new relation have unique names, even though the original relations might have attribute names in common.

Figure 9.8 The SELECT operation

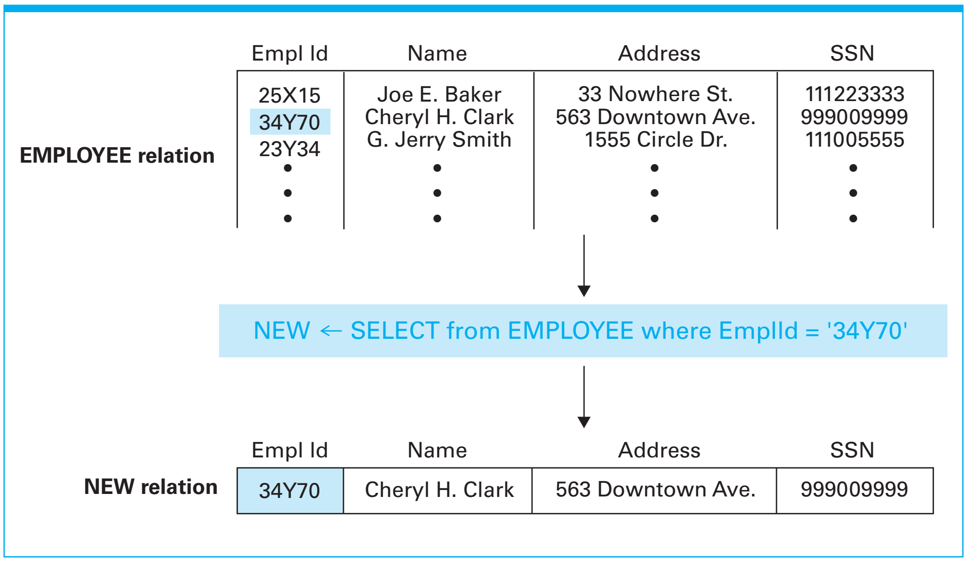
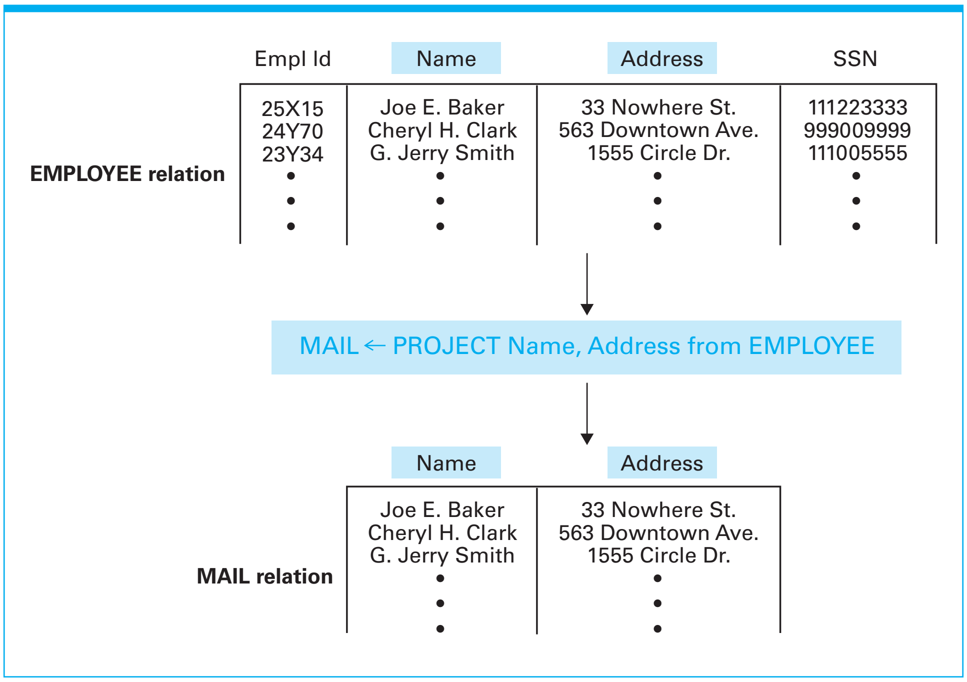
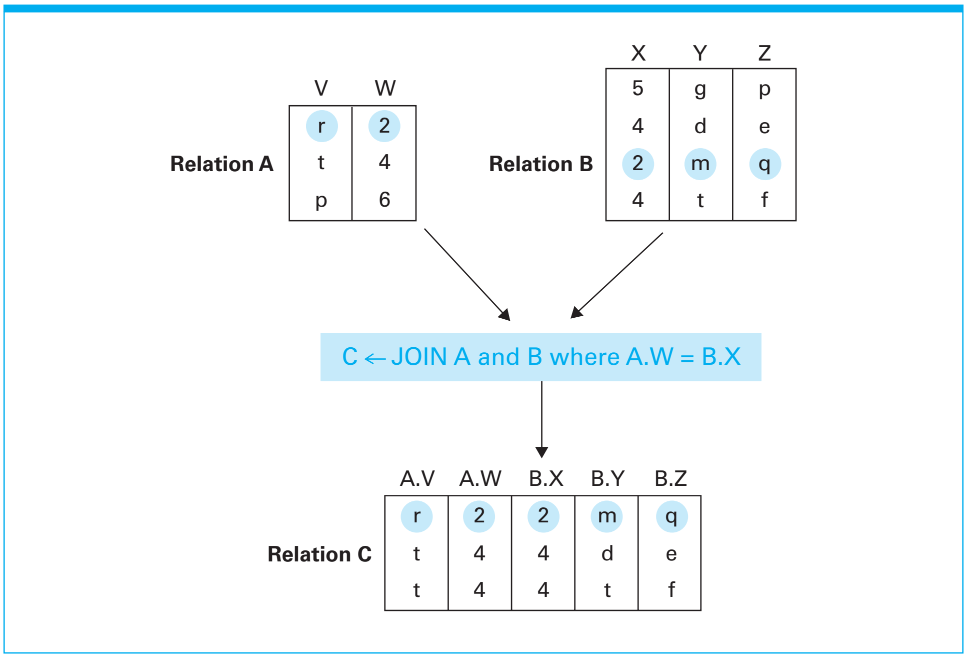


Figure 9.9 The PROJECT operation



The tuples (rows) of the new relation are produced by concatenating tuples from the two original relations (see again Figure 9.10). Which tuples are actually joined to form tuples in the new relation is determined by the condition under which the `JOIN` is constructed. One such condition is that designated attributes

Figure 9.10 The JOIN operation



have the same value. This, in fact, is the case represented in Figure 9.10, where we demonstrate the result of executing the statement

```
C ← JOIN A and B where A.W = B.X
```

In this example, a tuple from relation A should be concatenated with a tuple from relation B in exactly those cases where the attributes W and X in the two tuples are equal. Thus the concatenation of the tuple (r, 2) from relation A with the tuple (2, m, q) from relation B appears in the result because the value of attribute W in the first equals the value of attribute X in the second. On the other hand, the result of concatenating the tuple (r, 2) from relation A with the tuple (5, g, p) from relation B does not appear in the final relation because these tuples do not share common values in attributes W and X.

As another example, Figure 9.11 represents the result of executing the statement

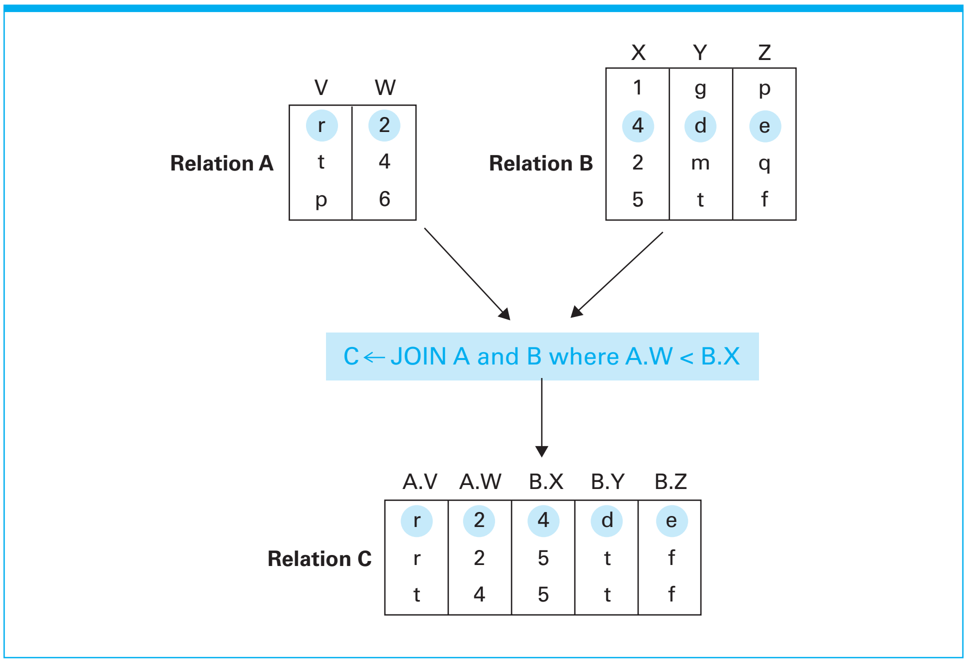
```
C ← JOIN A and B where A.W < B.X
```

Note that the tuples in the result are exactly those in which attribute W in relation A is less than attribute X in relation B.

Let us now see how the JOIN operation can be used with the database of Figure 9.5 to obtain a listing of all employee identification numbers along with the department in which each employee works. Our first observation is that the information required is distributed over more than one relation, and thus the process of retrieving the information must entail more than SELECTions and PROJECTions. In fact, the tool we need is the statement

```
NEW1 ← JOIN ASSIGNMENT and JOB
  where ASSIGNMENT.JobId = JOB.JobId
```

Figure 9.11 Another example of the JOIN operation



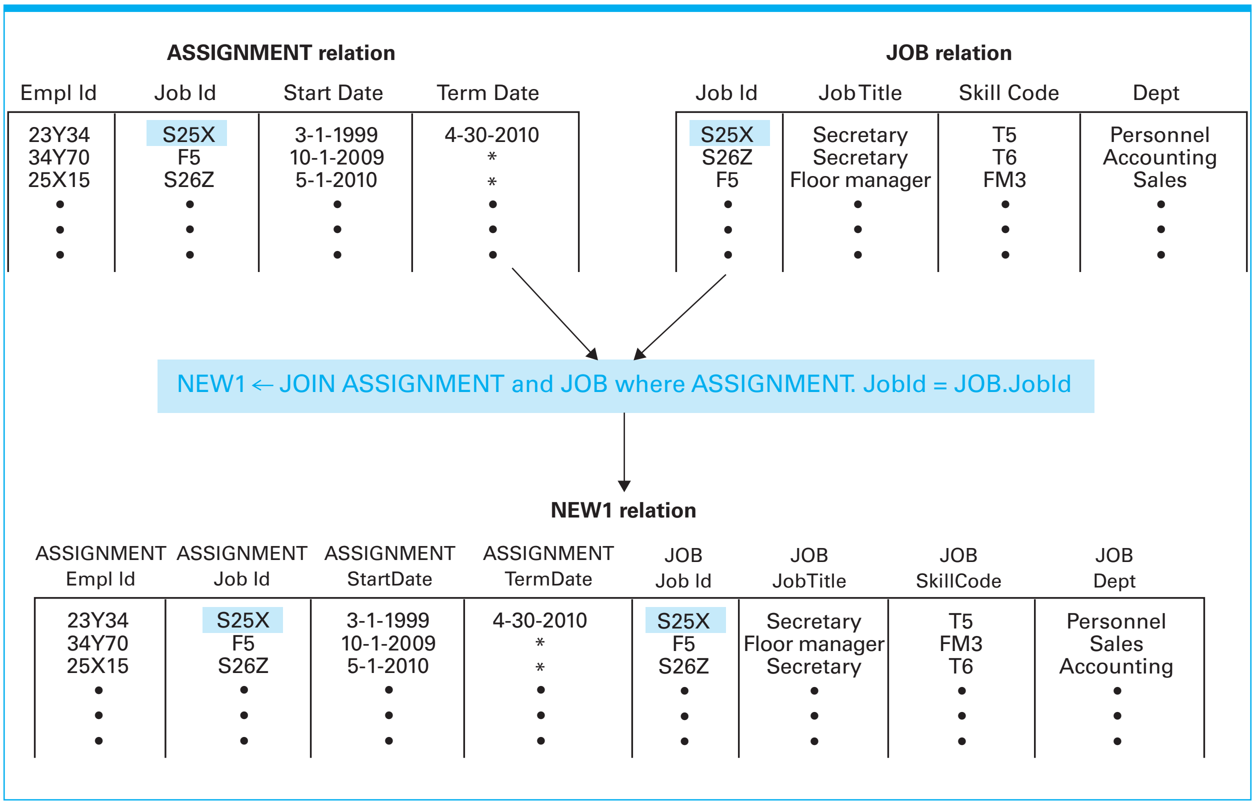
that produces the relation NEW1, as shown in Figure 9.12. From this relation, our problem can be solved by first SELECTing those tuples in which ASSIGNMENT.TermDate equals '*' (which indicates "still employed") and then PROJECTing the attributes ASSIGNMENT.EmplId and JOB.Dept. In short, the information we need can be obtained from the database in Figure 9.5 by executing the sequence

```
NEW1 ← JOIN ASSIGNMENT and JOB
  where ASSIGNMENT.JobId = JOB.JobId
NEW2 ← SELECT from NEW1 where ASSIGNMENT.TermDate = '*'
LIST ← PROJECT ASSIGNMENT.EmplId, JOB.Dept from NEW2
```

SQL

Now that we have introduced the basic relational operations, let us reconsider the overall structure of a database system. Remember that a database is actually stored in a mass storage system. To relieve the application programmer from the details of such systems, a database management system is provided that allows the application software to be written in terms of a database model, such as the relational model. The DBMS accepts commands in terms of the model and converts them into actions relative to the actual storage structure. This conversion is handled by a collection of routines within the DBMS that are used by the application software as abstract tools. Thus a DBMS based on the relational model would include routines to perform the SELECT, PROJECT, and JOIN operations, which

Figure 9.12 An application of the JOIN operation


NEW1 ← JOIN ASSIGNMENT and JOB where ASSIGNMENT. JobId = JOB.JobId

ASSIGNMENT relation

Empl Id	Job Id	Start Date	Term Date
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
25X15	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

JOB relation

Job Id	Job Title	Skill Code	Dept
S25X	Secretary	T5	Personnel
S26Z	Secretary	T6	Accounting
F5	Floor manager	FM3	Sales
•	•	•	•
•	•	•	•
•	•	•	•

NEW1 relation

ASSIGNMENT Empl Id	ASSIGNMENT Job Id	ASSIGNMENT StartDate	ASSIGNMENT TermDate	JOB Job Id	JOB JobTitle	JOB SkillCode	JOB Dept
23Y34	S25X	3-1-1999	4-30-2010	S25X	Secretary	T5	Personnel
34Y70	F5	10-1-2009	*	F5	Floor manager	FM3	Sales
25X15	S26Z	5-1-2010	*	S26Z	Secretary	T6	Accounting
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•

could then be called from the application software. In this manner the application software can be written as though the database were actually stored in the simple tabular form of the relational model.

Today's relational database management systems do not necessarily provide routines to perform the SELECT, PROJECT, and JOIN operations in their raw form. Instead, they provide routines that might be combinations of these basic steps. An example is the language SQL (Structured Query Language), which forms the backbone of most relational database query systems. For example, SQL is the underlying language in the relational database system MySQL (pronounced "My–S–Q–L") used by many database servers in the Internet.

One reason for SQL's popularity is that the American National Standards Institute has standardized it. Another reason is that it was originally developed and marketed by IBM and has thus benefited from a high level of exposure. In this section we explain how relational database queries are expressed in SQL.

Although we are about to find that a query stated in SQL is expressed in an imperative-sounding form, the reality is that it is essentially a declarative statement. You should read an SQL statement as a description of the information desired rather than a sequence of activities to be performed. The significance of this is that SQL relieves application programmers from the burden of developing algorithms for manipulating relations—they need merely to describe the information desired.

For our first example of an SQL statement, let us reconsider our last query in which we developed a three-step process for obtaining all employee identification numbers along with their corresponding departments. In SQL this entire query could be represented by the single statement

```
SELECT EmplId, Dept
FROM Assignment, Job
WHERE Assignment.JobId = Job.JobId
  AND Assignment.TermDate = '*';
```

As indicated by this example, each SQL query statement can contain three clauses: a select clause, a from clause, and a where clause. Roughly speaking, such a statement is a request for the result of forming the JOIN of all the relations listed in the from clause, SELECTing those tuples that satisfy the conditions in the where clause, and then PROJECTing those attributes listed in the select clause. (Note that the terminology is somewhat reversed since the select clause in an SQL statement identifies the attributes used in the PROJECT operation.) Let us consider some simple examples.

The statement

```
SELECT Name, Address
FROM Employee;
```

produces a listing of all employee names and addresses contained in the relation Employee. Note that this is merely a PROJECT operation.

The statement

```
SELECT EmplId, Name, Address, SSNum
FROM Employee
WHERE Name = 'Cheryl H. Clark';
```

produces all the information from the tuple associated with Cheryl H. Clark in the Employee relation. This is essentially a SELECT operation.

The statement

```
SELECT Name, Address
FROM Employee
WHERE Name = 'Cheryl H. Clark';
```

produces the name and address of Cheryl H. Clark as contained in the `Employee` relation. This is a combination of `SELECT` and `PROJECT` operations.

The statement

```
SELECT Employee.Name, Assignment.StartDate
FROM Employee, Assignment
WHERE Employee.EmplId = Assignment.EmplId;
```

produces a listing of all employee names and their dates of initial employment. Note that this is the result of `JOIN`ing the relations `Employee` and `Assignment` and then `SELECT`ing and `PROJECT`ing the appropriate tuples and attributes as identified in the where and select clauses.

We close by noting that SQL encompasses statements for defining the structure of relations, creating relations, and modifying the contents of relations as well as performing queries. For example, the following are examples of the `INSERT INTO`, `DELETE FROM`, and `UPDATE` statements.

The statement

```
INSERT INTO Employee
VALUES ('42Z12', 'Sue A. Burt', '33 Fair St.', '444661111');
```

adds a tuple to the `Employee` relation containing the values given;

```
DELETE FROM Employee
WHERE Name = 'G. Jerry Smith';
```

removes the tuple relating to G. Jerry Smith from the `Employee` relation; and

```
UPDATE Employee
SET Address = '1812 Napoleon Ave.'
WHERE Name = 'Joe E. Baker';
```

changes the address in the tuple associated with Joe E. Baker in the `Employee` relation.

Questions & Exercises

1. Answer the following questions based on the partial information given in the `EMPLOYEE`, `JOB`, and `ASSIGNMENT` relations in Figure 9.5:
   a. Who is the secretary in the accounting department with experience in the personnel department?
   b. Who is the floor manager in the sales department?
   c. What job does G. Jerry Smith currently hold?
2. Based on the `EMPLOYEE`, `JOB`, and `ASSIGNMENT` relations presented in Figure 9.5, write a sequence of relational operations to obtain a list of all job titles within the personnel department.

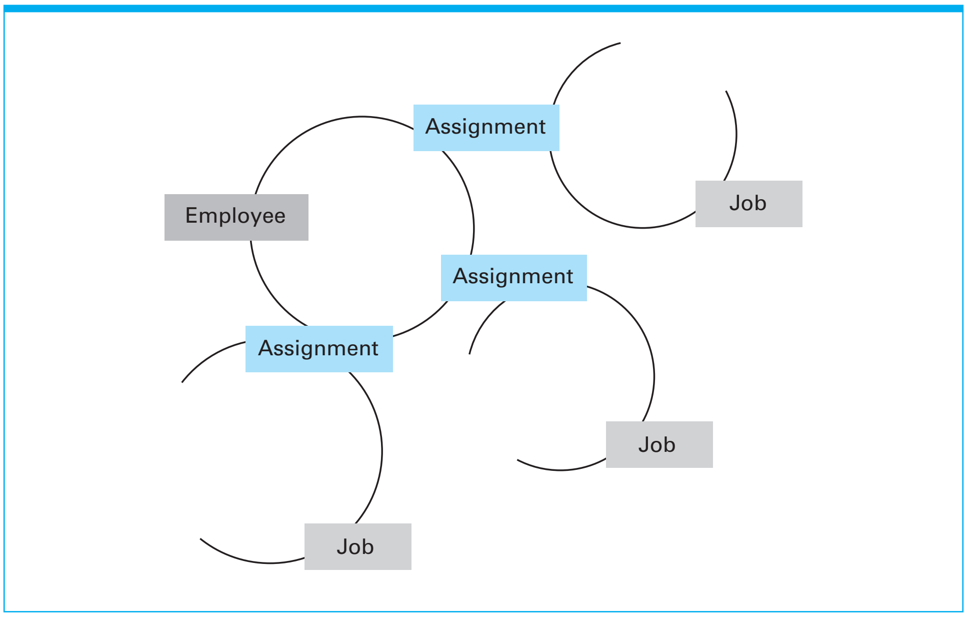
3. Based on the EMPLOYEE, JOB, and ASSIGNMENT relations presented in Figure 9.5, write a sequence of relational operations to obtain a list of employee names along with the employees' departments.
4. Convert your answers to questions 2 and 3 into SQL.
5. How does the relational model provide for data independence?
6. How are the different relations in a relational database tied together?

9.3 Object-Oriented Databases

Another database model is based on the object-oriented paradigm. This approach leads to an object-oriented database consisting of objects that are linked to each other to reflect their relationships. For example, an object-oriented implementation of the employee database from the previous section could consist of three classes (types of objects): `Employee`, `Job`, and `Assignment`. An object from the `Employee` class could contain such entries as `EmplId`, `Name`, `Address`, and `SSNum`; an object from the class `Job` could contain such entries as `JobId`, `JobTitle`, `SkillCode`, and `Dept`; and each object from the class `Assignment` could contain entries such as `StartDate` and `TermDate`.

A conceptual representation of such a database is shown in Figure 9.13 where the links between the various objects are represented by lines connecting the related objects. If we focus on an object of type `Employee`, we find it linked to a collection of objects of type `Assignment` representing the various assignments that that particular employee has held. In turn, each of these objects of type `Assignment` is linked to an object of type `Job` representing the job associated

Figure 9.13 The associations between objects in an object-oriented database



with that assignment. Thus all the assignments of an employee can be found by following the links from the object representing that employee. Similarly, all the employees who have held a particular job can be found by following the links from the object representing that job.

The links between objects in an object-oriented database are normally maintained by the DBMS, so the details of how these links are implemented are not a concern of the programmer writing application software. Instead, when a new object is added to the database, the application software merely specifies the other objects to which it should be linked. The DBMS then creates any linkage system that might be required to record these associations. In particular, a DBMS might link the objects representing the assignments of a given employee in a manner similar to a linked list.

Another task of an object-oriented DBMS is to provide permanent storage for the objects entrusted to it—a requirement that might seem obvious but is inherently distinct from the manner in which objects are normally treated. Normally, when an object-oriented program is executed, the objects created during the program's execution are discarded when the program terminates. In this sense the objects are considered transient. But objects that are created and added to a database must be saved after the program that created them terminates. Such objects are said to be **persistent.** Thus creating persistent objects is a significant departure from the norm.

Proponents of object-oriented databases offer numerous arguments to show why the object-oriented approach to database design is better than the relational approach. One is that the object-oriented approach allows the entire software system (application software, DBMS, and the database itself) to be designed in the same paradigm. This is in contrast to the historically common practice of using an imperative programming language to develop application software for interrogating a relational database. Inherent in such a task is the clash between imperative and relational paradigms. This distinction is subtle at our level of study but the difference has been the source of many software errors over the years. Even at our level, we can appreciate that an object-oriented database combined with an objected-oriented application program produces a homogeneous image of objects communicating with each other throughout the system. On the other hand, a relational database combined with an imperative application program conjures an image of two inherently different organizations trying to find a common interface.

To appreciate another advantage that object-oriented databases have over their relational counterparts, consider the problem of storing employee names in a relational database. If an entire name is stored as a single attribute in a relation, then inquiries regarding only surnames are awkward. However, if the name is stored as individual attributes, such as first name, middle name, and surname, then the number of attributes becomes problematic because not all names conform to a specific structure—even when the population is restricted to a single culture. In an object-oriented database, these issues can be hidden within the object that holds the employee's name. An employee's name can be stored as an intelligent object that is capable of reporting the related employee's name in a variety of formats. Thus, from outside these objects, it would be just as easy to deal with only surnames as with entire names, maiden names, or nicknames. The details involved with each perspective would be encapsulated within the objects.

This ability to encapsulate the technicalities of different data formats is advantageous in other cases as well. In a relational database, the attributes in a relation are part of the overall design of the database, and thus the types associated with these attributes permeate the entire DBMS. (Variables for temporary storage must be declared to be the appropriate type, and functions for manipulating data of the various types must be designed.) Thus, extending a relational database to include attributes of new types (audio and video) can be problematic. In particular, a variety of functions throughout the database design might need to be expanded to incorporate these new data types. In an object-oriented design, however, the same functions used to retrieve an object representing an employee's name can be used to retrieve an object representing a motion picture because the distinctions in type can be hidden within the objects involved. Thus the object-oriented approach appears to be more compatible with the construction of multimedia databases—a feature that is already proving to be a great advantage.

Still another advantage the object-oriented paradigm offers to database design is the potential for storing intelligent objects rather than merely data. That is, an object can contain methods describing how it should respond to messages regarding its contents and relationships. For example, each object of the class `Employee` in Figure 9.13 could contain methods for reporting and updating the information in the object as well as a method for reporting that employee's job history and perhaps a method for changing that employee's job assignment. Likewise, each object from the `Job` class could have a method for reporting the specifics of the job and perhaps a method for reporting those employees who have held that particular job. Thus to retrieve an employee's job history, we would not need to construct an elaborate function. Instead, we could merely ask the appropriate employee object to report its job history. This ability to construct databases whose components respond intelligently to inquiries offers an exciting array of possibilities beyond those of more traditional relational databases.

Questions & Exercises

1. What methods would be contained in an instance of an object from the `Assignment` class in the employee database discussed in this section?
2. What is a persistent object?
3. Identify some classes, as well as some of their internal characteristics, that can be used in an object-oriented database dealing with a warehouse inventory.
4. Identify an advantage that an object-oriented database can have over a relational database.

9.4 Maintaining Database Integrity

Inexpensive database management systems for personal use are relatively simple systems. They tend to have a single objective—to shield the user from the technical details of the database implementation. The databases maintained by these systems are relatively small and generally contain information whose loss or corruption would be inconvenient rather than disastrous. When a problem does arise, the user can usually correct the erroneous items directly or reload the database from a backup copy and manually make the modifications required to bring that copy up to date. This process might be inconvenient, but the cost of avoiding the inconvenience tends to be greater than the inconvenience itself. In any case, the inconvenience is restricted to only a few people, and any financial loss is generally limited.

In the case of large, multiuser, commercial database systems, however, the stakes are much higher. The cost of incorrect or lost data can be enormous and can have devastating consequences. In these environments, a major role of the DBMS is to maintain the database's integrity by guarding against problems such as operations that for some reason are only partially completed or different operations that might interact inadvertently to cause inaccurate information in the database. It is this role of a DBMS that we address in this section.

The Commit/Rollback Protocol

A single transaction, such as the transfer of funds from one bank account to another, the cancellation of an airline reservation, or the registration of a student in a university course, might involve multiple steps at the database level. For example, a transfer of funds between bank accounts requires that the balance in one account be decremented and the balance in the other be incremented. Between such steps the information in the database might be inconsistent. Indeed, funds are missing during the brief period after the first account has been decremented but before the other has been incremented. Likewise, when reassigning a passenger's seat on a flight, there might be an instant when the passenger has no seat or an instant when the passenger list appears to be one passenger greater than it actually is.

In the case of large databases that are subject to heavy transaction loads, it is highly likely that a random snapshot will find the database in the middle of some transaction. A request for the execution of a transaction or an equipment malfunction will therefore likely occur at a time when the database is in an inconsistent state.

Let us first consider the problem of a malfunction. The goal of the DBMS is to ensure that such a problem will not freeze the database in an inconsistent state. This is often accomplished by maintaining a log containing a record of each transaction's activities in a nonvolatile storage system, such as a magnetic disk. Before a transaction is allowed to alter the database, the alteration to be performed is first recorded in the log. Thus the log contains a permanent record of each transaction's actions.

The point at which all the steps in a transaction have been recorded in the log is called the **commit point.** It is at this point that the DBMS has the information it needs to reconstruct the transaction on its own if that should become necessary. At this point the DBMS becomes committed to the transaction in the sense that it accepts the responsibility of guaranteeing that the transaction's activities will be reflected in the database. In the case of an equipment malfunction, the DBMS can use the information in its log to reconstruct the transactions that have been completed (committed) since the last backup was made.

If problems should arise before a transaction has reached its commit point, the DBMS might find itself with a partially executed transaction that cannot be completed. In this case the log can be used to **roll back** (undo) the activities actually performed by the transaction. In the case of a malfunction, for instance, the DBMS could recover by rolling back those transactions that were incomplete (noncommitted) at the time of the malfunction.

Rollbacks of transactions are not restricted, however, to the process of recovering from equipment malfunctions. They are often a part of a DBMS's normal operation. For example, a transaction might be terminated before it has completed all its steps because of an attempt to access privileged information, or it might be involved in a deadlock in which competing transactions find themselves waiting for data being used by each other. In these cases, the DBMS can use the log to roll back a transaction and thus avoid an erroneous database due to incomplete transactions.

To emphasize the delicate nature of DBMS design, we should note that there are subtle problems lurking within the rollback process. The rolling back of one transaction might affect database entries that have been used by other transactions. For example, the transaction being rolled back might have updated an account balance, and another transaction might have already based its activities on this updated value. This might mean that these additional transactions must also be rolled back, which might adversely affect still other transactions. The result is the problem known as **cascading rollback.**

Locking

We now consider the problem of a transaction being executed while the database is in a state of flux from another transaction, a situation that can lead to inadvertent interaction between the transactions and produce erroneous results. For instance, the problem known as the **incorrect summary problem** can arise if one transaction is in the middle of transferring funds from one account to another when another transaction tries to compute the total deposits in the bank. This could result in a total that is either too large or too small depending on the order in which the transfer steps are performed. Another possibility is known as the **lost update problem,** which is exemplified by two transactions, each of which makes a deduction from the same account. If one transaction reads the account's current balance at the point when the other has just read the balance but has not yet calculated the new balance, then both transactions will base their deductions on the same initial balance. In turn, the effect of one of the deductions will not be reflected in the database.

To solve such problems, a DBMS could force transactions to execute in their entirety on a one-at-a-time basis by holding each new transaction in a queue until those preceding it have completed. But a transaction often spends a lot of time waiting for mass storage operations to be performed. By interweaving the execution of transactions, the time during which one transaction is waiting can be used by another transaction to process data it has already retrieved. Most large database management systems therefore contain a scheduler to coordinate time-sharing among transactions in much the same way that a multiprogramming operating system coordinates interweaving of processes (Section 3.3).

To guard against such anomalies as the incorrect summary problem and the lost update problem, these schedulers incorporate a **locking protocol** in which the items within a database that are currently being used by some transaction are marked as such. These marks are called locks; marked items are said to be locked. Two types of locks are common—**shared locks** and **exclusive locks.** They correspond to the two types of access to data that a transaction might require—shared access and exclusive access. If a transaction is not going to alter a data item, then it requires shared access, meaning that other transactions are also allowed to view the data. However, if the transaction is going to alter the item, it must have exclusive access, meaning that it must be the only transaction with access to that data.

In a locking protocol, each time a transaction requests access to a data item, it must also tell the DBMS the type of access it requires. If a transaction requests shared access to an item that is either unlocked or locked with a shared lock, that access is granted and the item is marked with a shared lock. If, however, the requested item is already marked with an exclusive lock, the additional access is denied. If a transaction requests exclusive access to an item, that request is granted only if the item has no lock associated with it. In this manner, a transaction that is going to alter data protects that data from other transactions by obtaining exclusive access, whereas several transactions can share access to an item if none of them are going to change it. Of course, once a transaction is finished with an item, it notifies the DBMS, and the associated lock is removed.

Various algorithms are used to handle the case in which a transaction's access request is rejected. One algorithm is that the transaction is merely forced to wait until the requested item becomes available. This approach, however, can lead to deadlock, since two transactions that require exclusive access to the same two data items could block each other's progress if each obtains exclusive access to one of the items and then insists on waiting for the other. To avoid such deadlocks, some database management systems give priority to older transactions. That is, if an older transaction requires access to an item that is locked by a younger transaction, the younger transaction is forced to release all of its data items, and its activities are rolled back (based on the log). Then, the older transaction is given access to the item it required, and the younger transaction is forced to start again. If a younger transaction is repeatedly preempted, it will grow older in the process and ultimately become one of the older transactions with high priority. This protocol, known as the **wound-wait protocol** (old transactions wound young transactions, young transactions wait for old ones), ensures that every transaction will ultimately be allowed to complete its task.

Questions & Exercises

1. What is the difference between a transaction that has reached its commit point and one that has not?
2. How could a DBMS guard against extensive cascading rollback?
3. Show how the uncontrolled interweaving of two transactions, one deducting $100 from an account and the other deducting $200 from the same account, could produce final balances of $100, $200, and $300, assuming that the initial balance is $400.

4. a. Summarize the possible results of a transaction requesting shared access to an item in a database.
   b. Summarize the possible results of a transaction requesting exclusive access to an item in a database.
5. Describe a sequence of events that would lead to deadlock among transactions performing operations on a database system.
6. Describe how the deadlock in your answer to question 5 could be broken. Would your solution require use of the database management system's log? Explain your answer.

9.5 Traditional File Structures

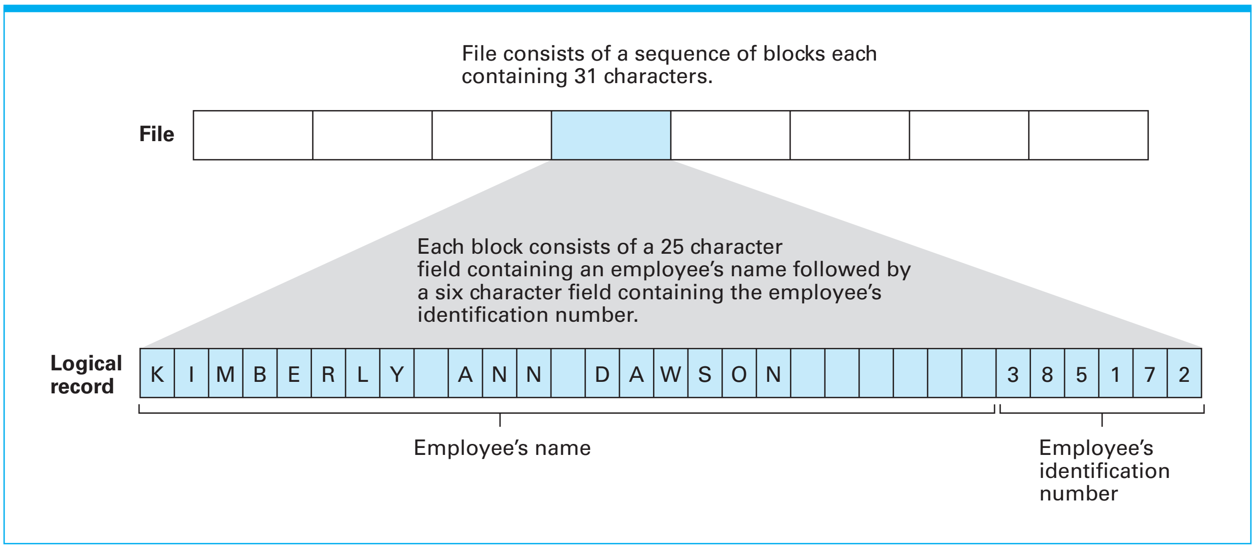
In this section we digress from our study of mutidimensional database systems to consider traditional file structures. These structures represent the historical beginning of data storage and retrieval systems from which current database technology has evolved. Many of the techniques developed for these structures (such as indexing and hashing) are important tools in the construction of today's massive, complex databases.

Sequential Files

A **sequential file** is a file that is accessed in a serial manner from its beginning to its end as though the information in the file were arranged in one long row. Examples include audio files, video files, files containing programs, and files containing textual documents. In fact, most of the files created by a typical personal computer user are sequential files. For instance, when a spreadsheet is saved, its information is encoded and stored as a sequential file from which the spreadsheet application software can reconstruct the spreadsheet.

Text files, which are sequential files in which each logical record is a single symbol encoded using ASCII or Unicode, often serve as a basic tool for constructing more elaborate sequential files such as an employee records file. One only needs to establish a uniform format for representing the information about each employee as a string of text, encode the information according to that format, and then record the resulting employee records one after another as one single string of text. For example, one could construct a simple employee file by agreeing to enter each employee record as a string of 31 characters, consisting of a field of 25 characters containing the employee's name (filled with enough blanks to complete the 25-character field), followed by a field of 6 characters representing the employee's identification number. The final file would be a long string of encoded characters in which each 31-character block represents the information about a single employee (Figure 9.14). Information would be retrieved from the file in terms of logical records consisting of 31-character blocks. Within each of these blocks, individual fields would be identified according to the uniform format with which the blocks were constructed.

Figure 9.14 The structure of a simple employee file implemented as a text file


File consists of a sequence of blocks each containing 31 characters.
File
Each block consists of a 25 character field containing an employee's name followed by a six character field containing the employee's identification number.
Logical record
K I M B E R L Y A N N D A W S O N 3 8 5 1 7 2
Employee's name
Employee's identification number

The data in a sequential file must be recorded in mass storage in such a way that the sequential nature of the file is preserved. If the mass storage system is itself sequential (as in the case of a magnetic tape or CD), this is a straightforward undertaking. We need merely record the file on the storage medium according to the sequential properties of the medium. Then processing the file is the task of merely reading and processing the file's contents in the order in which they are found. This is exactly the process followed in playing audio CDs, where the music is stored as a sequential file sector by sector along one continuous spiraling track.

In the case of magnetic disk storage, however, the file would be scattered over different sectors that could be retrieved in a variety of orders. To preserve the proper order, most operating systems (more precisely, the file manager) maintain a list of the sectors on which the file is stored. This list is recorded as part of the disk's directory system on the same disk as the file. By means of this list, the operating system can retrieve the sectors in the proper sequence as though the file were stored sequentially, even though the file is actually distributed over various portions of the disk.

Inherent in processing a sequential file is the need to detect when the end of the file is reached. Generically, we refer to the end of a sequential file as the **end-of-file (EOF).** There are a variety of ways of identifying the EOF. One is to place a special record, called a **sentinel,** at the end of the file. Another is to use the information in the operating system's directory system to identify a file's EOF. That is, since the operating system knows which sectors contain the file, it also knows where the file terminates. A classic example involving sequential files is payroll processing in a small company. Here we imagine a sequential file consisting of a series of logical records, each of which contains the information about an employee's pay (name, employee identification number, pay scale, and so on) from which checks must be printed on a routine basis. As each employee record is retrieved, that employee's pay is calculated, and the appropriate check produced. The activity of processing such a sequential file is exemplified by the statement

```
while (the EOF has not been reached):
  retrieve the next record from the file and process it
```

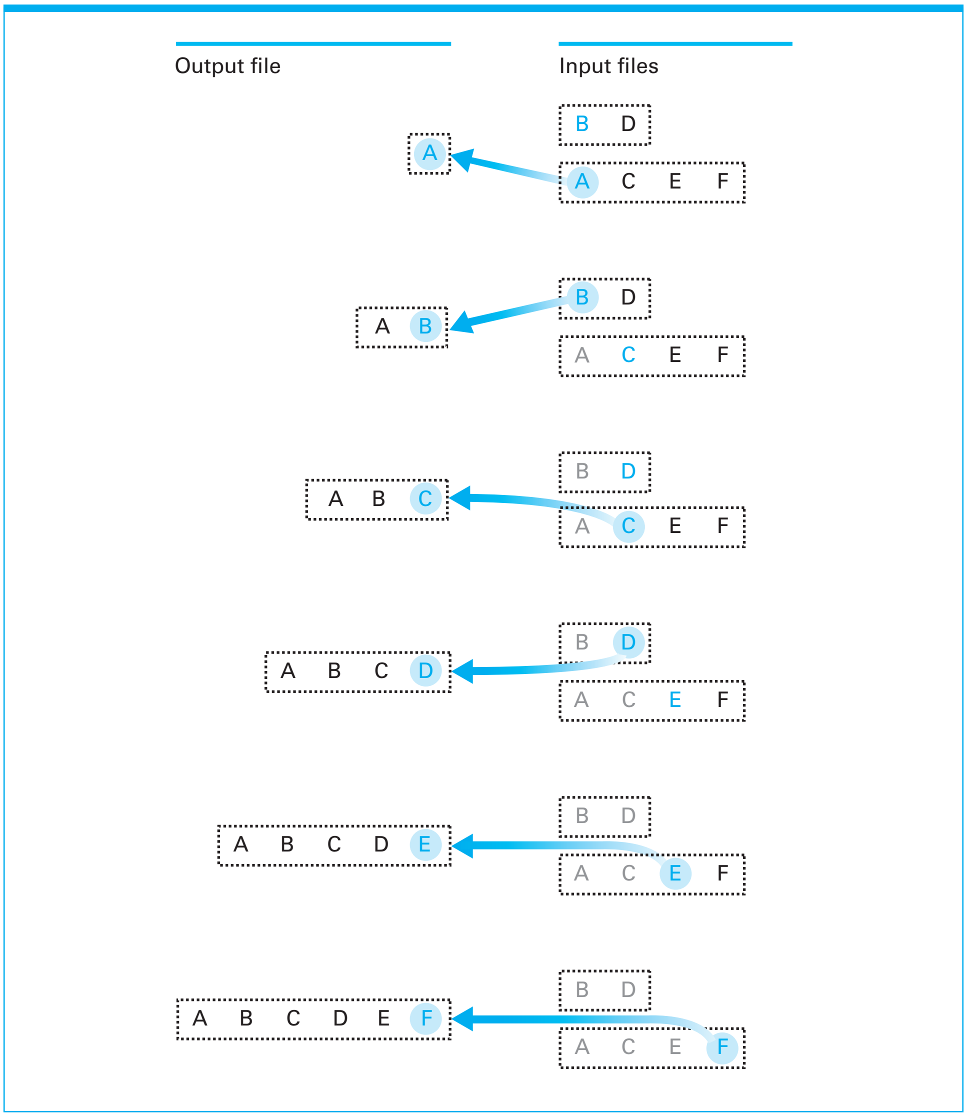
When the logical records within a sequential file are identified by key field values, the file is usually arranged so that the records appear in the order determined by the keys (perhaps alphabetical or numerical). Such an arrangement simplifies the task of processing the information in the file. For example, suppose that processing payroll requires that each employee record be updated to reflect the information on that employee's time sheet. If both the file containing the time sheet records and the file containing the employee records are in the same order according to the same keys, then this updating process can be handled by accessing both files sequentially—using the time sheet retrieved from one file to update the corresponding record from the other file. This is a significant improvement over the repeated searching process that would be required if the files were not in corresponding order. Thus updating classic sequential files is typically handled in multiple steps. First, the new information (such as the collection of time sheets) is recorded in a sequential file known as a transaction file, and this transaction file is sorted to match the order of the file to be updated, which is called the master file. Then, the records in the master file are updated by retrieving the records from both files sequentially.

A slight variation of this updating process is the process of merging two sequential files to form a new file containing the records from the two originals. The records in the input files are assumed to be arranged in ascending order according to a common key field, and it is also assumed that the files are to be merged in a manner that produces an output file whose keys are also in ascending order. The classic merge algorithm is summarized in Figure 9.15. The underlying theme is to build the output file as the two input files are scanned sequentially (Figure 9.16).

Figure 9.15 A function for merging two sequential files

```
def MergeFiles (InputFileA, InputFileB, OutputFile):
  if (both input files at EOF):
    Stop, with OutputFile empty.
  if (InputFileA not at EOF):
    Declare its first record to be its current record.
  if (InputFileB not at EOF):
    Declare its first record to be its current record.
  while (neither input file at EOF):
    Put the current record with the "smaller" key field
      value in OutputFile.
    if (that current record is the
      last record in its corresponding input file):
      Declare that input file to be at EOF.
    else:
      Declare the next record in that input file
        to be the file's current record.
  Starting with the current record in the input file that is not at
  EOF, copy the remaining records to OutputFile.
```

Figure 9.16 Applying the merge algorithm. (Letters are used to represent entire records. The particular letter indicates the value of the record's key field)



Indexed Files

Sequential files are ideal for storing data that will be processed in the order in which the file's entries are stored. However, such files are inefficient when records within the file must be retrieved in an unpredictable order. In such situations what is needed is a way to identify the location of the desired logical record quickly. A popular solution is to use an index for the file in much the same way that an index in a book is used to locate topics within the book. Such a file system is called an **indexed file.**

An index for a file contains a list of the keys stored in the file along with entries indicating where the record containing each key is stored. Thus to find a particular record, one finds the identifying key in the index and then retrieves the block of information stored at the location associated with that key.

A file's index is normally stored as a separate file on the same mass storage device as the indexed file. The index is usually transferred to main memory before file processing begins so that it is easily accessible when access to records in the file is required (Figure 9.17).

A classic example of an indexed file occurs in the context of maintaining employee records. Here an index can be used to avoid lengthy searches when you are retrieving an individual record. In particular, if the file of employee records is indexed by employee identification numbers, then an employee's record can be retrieved quickly when the employee's identification number is known. Another example is found on audio CDs where an index is used to allow relatively quick access to individual recordings.

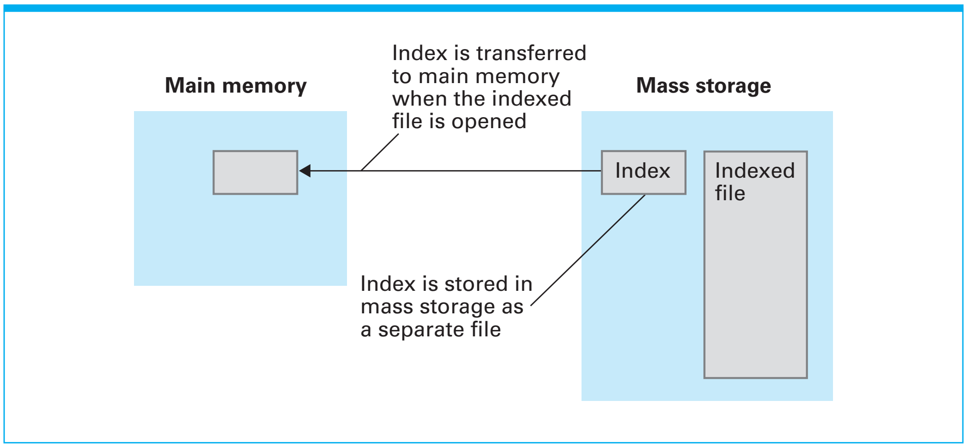
Over the years numerous variations of the basic index concept have been used. One variation constructs an index in a hierarchical manner so that the index takes on a layered or tree structure. A prominent example is the hierarchical directory system used by most operating systems for organizing file storage. In such a case, the directories, or folders, play the role of indexes, each containing links to its subindexes. From this perspective, the entire file system is merely one large indexed file.

Hash Files

Although indexing provides relatively quick access to entries within a data storage structure, it does so at the expense of index maintenance. **Hashing** is a technique that provides similar access without such overhead. As in the case of an indexed system, hashing allows a record to be located by means of a key value. But, rather than looking up the key in an index, hashing identifies the location of the record directly from the key.

A hash system can be summarized as follows: The data storage space is divided into several sections, called **buckets,** each of which is capable of holding several records. The records are dispersed among the buckets according to an algorithm that converts key values into bucket numbers. (This conversion from key values to bucket numbers is called a **hash function.**) Each record is stored in the bucket identified by this process. Therefore, a record that has been placed in the storage structure can be retrieved by first applying the hash function to the

Figure 9.17 Opening an indexed file


Main memory
Index is transferred to main memory when the indexed file is opened
Mass storage
Index
Indexed file
Index is stored in mass storage as a separate file

Authentication via Hashing

Hashing is much more than a means of constructing efficient data storage systems. For example, hashing can be used as a means of authenticating messages transferred over the Internet. The underlying theme is to hash the message in a secret way. This value is then transferred with the message. To authenticate the message, the receiver hashes the message received (in the same secret way) and confirms that the value produced agrees with the original value. (The odds of an altered message hashing to the same value are assumed to be very small.) If the value obtained does not agree with the original value, the message is exposed as being corrupted. Those who are interested might wish to search the Internet for information about MD5, which is a hash function used extensively in authentication applications.

It is enlightening to consider error detection techniques as an application of hashing for authentication. For example, the use of parity bits is essentially a hashing system in which a bit pattern is hashed to produce either a 0 or a 1. This value is then transferred along with the original pattern. If the pattern ultimately received does not hash to that same value, the pattern is considered corrupted.

record's identifying key to determine the appropriate bucket, then retrieving the contents of that bucket, and finally searching through the data retrieved for the desired record.

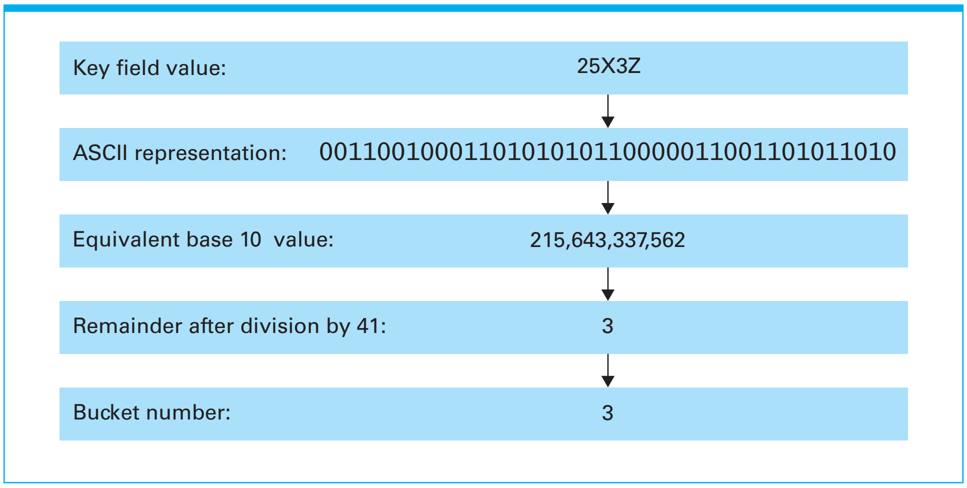
Hashing is not only used as a means of retrieving data from mass storage but also as a means of retrieving items from large blocks of data stored in main memory. When hashing is applied to a storage structure in mass storage, the result is called a **hash file.** When applied to a storage structure within main memory, the result is usually called a **hash table.**

Let us apply the hashing technique to the classic employee file in which each record contains information about a single employee in a company. First, we establish several available areas of mass storage that will play the role of buckets. The number of buckets and the size of each bucket are design decisions that we will consider later. For now, let us assume that we have created 41 buckets, which we refer to as bucket number 0, bucket number 1, through bucket number 40. (The reason we selected 41 buckets rather than an even 40 will be explained shortly.)

Let us assume that an employee's identification number will be used as the key for identifying the employee's record. Our next task, then, is to develop a hash function for converting these keys into bucket numbers. Although the employee identification "numbers" might have the form 25X3Z or J2X35 and are therefore not numeric, they are stored as bit patterns, and we can interpret the bit patterns as numbers. Using this numeric interpretation, we can divide any key by the number of buckets available and record the remainder, which in our case will be an integer in the range from 0 to 40. Thus we can use the remainder of this division process to identify one of the 41 buckets (Figure 9.18).

Using this as our hash function, we proceed to construct the file by considering each record individually, applying our divide-by-41 hash function to its key to obtain a bucket number, and then storing the record in that bucket (Figure 9.19). Later, if we need to retrieve a record, we need merely apply our hash function to the record's key to identify the appropriate bucket and then search that bucket for the record in question.

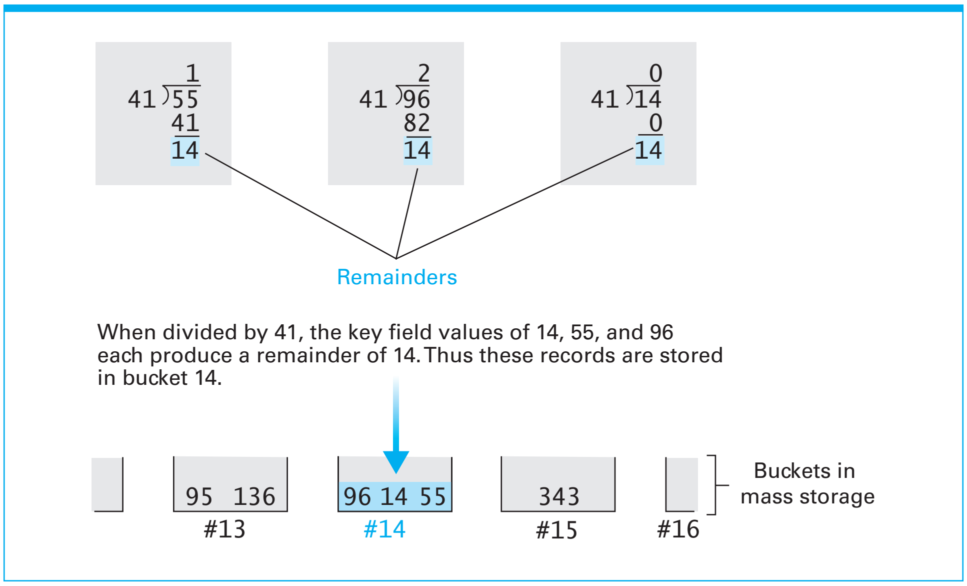
Figure 9.18 Hashing the key field value 25X3Z to one of 41 buckets


Key field value: 25X3Z
ASCII representation: 0011001000110101010110000011001101011010
Equivalent base 10 value: 215,643,337,562
Remainder after division by 41: 3
Bucket number: 3

At this point let us reconsider our decision to divide the storage area into 41 buckets. First, note that to obtain an efficient hash system, the records being stored should be distributed evenly among the buckets. If a disproportionate number of keys happen to hash to the same bucket (a phenomenon called **clustering**), then a disproportionate number of records will be stored in a single bucket. In turn, retrieving a record from that bucket could require a time-consuming search, causing the loss of any benefits gained by hashing.

Now observe that if we had chosen to divide the storage area into 40 buckets rather than 41, our hash function would have involved dividing the keys by the value 40 rather than 41. But, if a dividend and a divisor both have a common

Figure 9.19 The rudiments of a hashing system



When divided by 41, the key field values of 14, 55, and 96 each produce a remainder of 14. Thus these records are stored in bucket 14.

factor, that factor will be present in the remainder as well. In particular, if the keys to the entries stored in our hash file happened to be multiples of 5 (which is also a divisor of 40), then the factor of 5 would appear in the remainders when divided by 40, and the entries would cluster in those buckets associated with the remainders 0, 5, 10, 15, 20, 25, 30, and 35. Similar situations would occur in the case of keys that are multiples of 2, 4, 8, 10, and 20, because they are all also factors of 40. Consequently, we choose to divide the storage area into 41 buckets because the choice of 41, being a prime number, eliminated the possibility of common factors and therefore reduced the chance of clustering.

Unfortunately, the possibility of clustering can never be completely eliminated. Even with a well-designed hash function, it is highly likely that two keys will hash to the same value, a phenomenon called a collision, early in the file construction process. To understand why, consider the following scenario.

Suppose that we have found a hash function that arbitrarily distributes records among 41 buckets, that our storage system is empty, and that we are going to insert new records one at a time. When we insert the first record, it will be placed in an empty bucket. However, when we insert the next record, only 40 of the 41 buckets are still empty, so the probability that the second record will be placed in an empty bucket is only 40/41. Assuming that the second record is placed in an empty bucket, the third record finds only 39 empty buckets, and thus the probability of it being placed in one of them is 39/41. Continuing this process, we find that if the first seven records are placed in empty buckets, the eighth record has only a 34/41 probability of being placed in one of the remaining empty buckets.

This analysis allows us to compute the probability that all the first eight records will be placed in empty buckets—it is the product of the probabilities of each record being placed in an empty bucket, assuming that the preceding entries were so placed. This probability is

```
(41/41)(40/41)(39/41)(38/41) . . . (34/41) = .482
```

The point is that the result is less than one-half. That is, it is more likely than not that in distributing records among 41 buckets a collision will have occurred by the time the eighth record is stored.

The high probability of collisions indicates that, regardless of how well-chosen a hash function might be, any hash system must be designed with clustering in mind. In particular, it is possible that a bucket might fill up and overflow. One approach to this problem would be to allow the buckets to expand in size. Another approach is to allow buckets to spill into an overflow area that has been reserved for that purpose. In any case the occurrence of clustering and overflowing buckets can significantly degrade the performance of a hash file.

Research has shown that, as a general rule, hash files perform well as long as the ratio of the number of records to the total record capacity of the file (a ratio known as the **load factor**) remains below 50 percent. However, if the load factor begins to creep above 75 percent, the system's performance generally degrades (clustering raises its ugly head, causing some buckets to fill and possibly overflow). For this reason, a hash storage system is usually reconstructed with a larger capacity if its load factor approaches the 75 percent value. We conclude that the efficiency of record retrieval obtained by implementing a hash system is not gained without cost.

Questions & Exercises

1. Follow the merge algorithm presented in Figure 9.15, assuming that one input file contains records with key field values equal to B and E while the other contains A, C, D, and F.
2. The merge algorithm is the heart of a popular sort algorithm called the merge sort. Can you discover this algorithm? (Hint: Any nonempty file can be considered to be a collection of one-entry files.)
3. Is being sequential a physical or conceptual property of a file?
4. What are the steps required when retrieving a record from an indexed file?
5. Explain how a poorly chosen hash function can result in a hash storage system becoming little more than a sequential file.
6. Suppose a hash storage system is constructed using the division hash function as presented in the text but with six storage buckets. For each of the following key values, identify the bucket in which the record with that key is placed. What goes wrong and why?

a. 24	b. 30	c. 3	d. 18	e. 15
f. 21	g. 9	h. 39	i. 27	j. 0

7. How many people must be gathered together before the odds are that two members of the group will have birthdays on the same day of the year? How does this problem relate to the material in this section?

9.6 Data Mining

A rapidly expanding subject that is closely associated with database technology is data mining, which consists of techniques for discovering patterns in collections of data. Data mining has become an important tool in numerous areas including marketing, inventory management, quality control, loan risk management, fraud detection, and investment analysis. Data mining techniques even have applications in what might seem unlikely settings as exemplified by their use in identifying the functions of particular genes encoded in DNA molecules and characterizing properties of organisms.

Data mining activities differ from traditional database interrogation in that data mining seeks to identify previously unknown patterns as opposed to traditional database inquiries that merely ask for the retrieval of stored facts. Moreover, data mining is practiced on static data collections, called **data warehouses,** rather than “online” operational databases that are subject to frequent updates. These warehouses are often “snapshots” of databases or collections of databases. They are used in lieu of the actual operational databases because finding patterns in a static system is easier than in a dynamic one.

We should also note that the subject of data mining is not restricted to the domain of computing but has tentacles that extend far into statistics. In fact, many would argue that since data mining had its origins in attempts to perform statistical analysis on large, diverse data collections, it is an application of statistics rather than a field of computer science.

Two common forms of data mining are **class description** and **class discrimination.** Class description deals with identifying properties that characterize a given group of data items, whereas class discrimination deals with identifying properties that divide two groups. For example, class description techniques would be used to identify characteristics of people who buy small economical vehicles, whereas class discrimination techniques would be used to find properties that distinguish customers who shop for used cars from those who shop for new ones.

Another form of data mining is **cluster analysis,** which seeks to discover classes. Note that this differs from class description, which seeks to discover properties of members within classes that are already identified. More precisely, cluster analysis tries to find properties of data items that lead to the discovery of groupings. For example, in analyzing information about people's ages who have viewed a particular motion picture, cluster analysis might find that the customer base breaks down into two age groups—a 4-to-0 age group and a 25-to-40 age group. (Perhaps the motion picture attracted children and their parents?)

Still another form of data mining is **association analysis,** which involves looking for links between data groups. It is association analysis that might reveal that customers who buy potato chips also buy beer and soda or that people who shop during the traditional weekday work hours also draw retirement benefits.

Outlier analysis is another form of data mining. It tries to identify data entries that do not comply with the norm. Outlier analysis can be used to identify errors in data collections, to identify credit card theft by detecting sudden deviations from a customer's normal purchase patterns, and perhaps to identify potential terrorists by recognizing unusual behavior.

Finally, the form of data mining called **sequential pattern analysis** tries to identify patterns of behavior over time. For example, sequential pattern analysis might reveal trends in economic systems such as equity markets or in environmental systems such as climate conditions.

As indicated by our last example, results from data mining can be used to predict future behavior. If an entity possesses the properties that characterize a class, then the entity will probably behave like members of that class. However, many data mining projects are aimed at merely gaining a better understanding

Bioinformatics

Advances in database technology and data mining techniques are expanding the repertoire of tools available to biologists in research areas involving the identification of patterns and the classification of organic compounds. The result is a new field within biology called bioinformatics. Having originated in endeavors to decode DNA, bioinformatics now encompasses such tasks as cataloguing proteins and understanding sequences of protein interactions (called biochemical pathways). Although normally considered to be a part of biology, bioinformatics is an example of how computer science is influencing and even becoming ingrained in other fields.

of the data, as witnessed by the use of data mining in unraveling the mysteries of DNA. In any case, the scope of data mining applications is potentially enormous, and thus data mining promises to be an active area of research for years to come.

Note that database technology and data mining are close cousins, and thus research in one will have repercussions in the other. Database techniques are used extensively to give data warehouses the capability of presenting data in the form of **data cubes** (data viewed from multiple perspectives—the term *cube* is used to conjecture the image of multiple dimensions) that make data mining possible. In turn, as researchers in data mining improve techniques for implementing data cubes, these results will pay dividends in the field of database design.

In closing, we should recognize that successful data mining encompasses much more than the identification of patterns within a collection of data. Intelligent judgment must be applied to determine whether those patterns are significant or merely coincidences. The fact that a particular convenience store has sold a high number of winning lottery tickets should probably not be considered significant to someone planning to buy a lottery ticket, but the discovery that customers who buy snack food also tend to buy frozen dinners might constitute meaningful information to a grocery store manager. Likewise, data mining encompasses a vast number of ethical issues involving the rights of individuals represented in the data warehouse, the accuracy and use of the conclusions drawn, and even the appropriateness of data mining in the first place.

Questions & Exercises

1. Why is data mining not conducted on “online” databases?
2. Give an additional example of a pattern that might be found by each of the types of data mining identified in the text.
3. Identify some different perspectives that a data cube might allow when mining sales data.
4. How does data mining differ from traditional database inquiries?

9.7 Social Impact of Database Technology

With the development of database technology, information that was once buried in arcane records has become accessible. In many cases, automated library systems place a patron’s reading habits within easy reach, retailers maintain records of their customers’ purchases, and Internet search engines keep records of their clients’ requests. In turn this information is potentially available to marketing firms, law enforcement agencies, political parties, employers, and private individuals.

This is representative of the potential problems that permeate the entire spectrum of database applications. Technology has made it easy to collect enormous amounts of data and to merge or compare different data collections to obtain relationships that would otherwise remain buried in the heap. The ramifications, both positive and negative, are enormous. These ramifications are not merely subjects for academic debate—they are realities.

In some cases the data collection process is readily apparent; in others it is subtle. Examples of the first case occur when one is explicitly asked to provide information. This may be done in a voluntary manner, as in surveys or contest registration forms, or it may be done in an involuntary manner, such as when imposed by government regulations. Sometimes whether it is voluntary depends on one's point of view. Is providing personal information when applying for a loan voluntary or involuntary? The distinction depends on whether receiving the loan is a convenience or a necessity. To use a credit card at some retailers now requires that you allow your signature to be recorded in a digitized format. Again, providing the information is either voluntary or involuntary depending on your situation.

More subtle cases of data collection avoid direct communication with the subject. Examples include a credit company that records the purchasing practices of the holders of its credit cards, websites that record the identities of those who visit the site, and social activists who record the license plate numbers on the cars parked in a targeted institution's parking lot. In these cases the subject of the data collection might not be aware that information is being collected and less likely to be aware of the existence of the databases being constructed.

Sometimes the underlying data-collection activities are self-evident if one merely stops to think. For example, a grocery store might offer discounts to its regular customers who register in advance with the store. The registration process might involve the issuance of identification cards that must be presented at the time of purchase to obtain the discount. The result is that the store is able to compile a record of the customers' purchases—a record whose value far exceeds the value of the discounts awarded.

Of course, the force driving this boom in data collection is the value of the data, which is amplified by advances in database technology that allow data to be linked in ways that reveal information that would otherwise remain obscure. For example, the purchasing patterns of credit card holders can be classified and cross-listed to obtain customer profiles of immense marketing value. Subscription forms for body-building magazines can be mailed to those who have recently purchased exercise equipment, whereas subscription forms for dog obedience magazines can be targeted toward those who have recently purchased dog food. Alternative ways of combining information are sometimes very imaginative. Welfare records have been compared to criminal records to find and apprehend parole violators, and in 1984 the Selective Service in the United States used old birthday registration lists from a popular ice cream retailer to identify citizens who had failed to register for the military draft.

There are several approaches to protecting society from abusive use of databases. One is to apply legal remedies. Unfortunately, passing a law against an action does not stop the action from occurring but merely makes the action illegal. A prime example in the United States is the Privacy Act of 1974 whose purpose was to protect citizens from abusive use of government databases. One provision of this act required government agencies to publish notice of their databases in the Federal Register to allow citizens to access and correct their personal information. However, government agencies were slow to comply with this provision. This does not necessarily imply malicious intent. In many cases the problem was one of bureaucracy. But, the fact that a bureaucracy might be constructing personnel databases that it is unable to identify is not reassuring.

Another, and perhaps more powerful, approach to controlling database abuse is public opinion. Databases will not be abused if the penalties outweigh the benefits; and the penalty businesses fear the most is adverse public opinion—this goes right to the bottom line. In the early 1990s it was public opinion that ultimately stopped major credit bureaus from selling mailing lists for marketing purposes. More recently, Google discontinued its Google Buzz social networking tool in 2011 after its automatic sharing of contact information from the popular Gmail tool was greeted with harsh public criticism. Even government agencies have bowed to public opinion. In 1997 the Social Security Administration in the United States modified its plan to make Social Security records available via the Internet when public opinion questioned the security of the information. In some of these cases results were obtained in days—a stark contrast to the extended time periods associated with legal processes.

Of course, in many cases database applications are beneficial to both the holder and the subject of the data, but in all cases there is a loss of privacy that should not be taken lightly. Such privacy issues are serious when the information is accurate, but they become gigantic when the information is erroneous. Imagine the feeling of hopelessness if you realized that your credit rating was adversely affected by erroneous information. Imagine how your problems would be amplified in an environment in which this misinformation was readily shared with other institutions. Privacy problems are, and will be, a major side effect of advancing technology in general and database techniques in particular. The solutions to these problems will require an educated, alert, and active citizenry.

Questions & Exercises

1. Should law enforcement agencies be given access to databases for the purpose of identifying individuals with criminal tendencies, even though the individuals might not have committed a crime?
2. Should insurance companies be given access to databases for the purpose of identifying individuals with potential medical problems, even though the individuals have not shown any symptoms?
3. Suppose you were financially comfortable. What benefits could you derive if this information were shared among a variety of institutions? What penalties could you suffer from the distribution of this same information? What if you were financially uncomfortable?
4. What role does a free press have in controlling database abuse? (For example, to what extent does the press affect public opinion or expose abuse?)

Chapter Review Problems

(Asterisked problems are associated with optional sections.)

1. What is the significance of database management systems?
2. What is a database model?
3. What is a lossless decomposition or a nonloss decomposition?
4. What is the difference between a tuple and an attribute?
5. Identify two benefits of separating application software from the DBMS.
6. Describe the similarities between an abstract data type (Chapter 8) and a database model.
7. Identify the level within a database system (user, programmer of application software, designer of the DBMS software) at which each of the following concerns or activities occur:
   a. How should data be stored on a disk to maximize efficiency?
   b. Is there a vacancy on flight 243?
   c. How should a relation be organized in mass storage?
   d. How many times should a user be allowed to mistype a password before the conversation is terminated?
   e. How can the PROJECT operation be implemented?
8. Which of the following tasks are handled by a DBMS?
   a. Ensure that a user's access to the database is restricted to the appropriate subschema.
   b. Translate commands stated in terms of the database model into actions compatible with the actual data storage system.
   c. Disguise the fact that the data in the database is actually scattered among many computers in a network.
9. Describe how the following information about railways, trains (for a particular day), and passengers would be represented in a relational database:

   Trains: Rolar, Omni, and Holiday

   Trains for Rolar: R221, R567, and R234

   Holiday special Trains: H897 and H008

   Trains for Omni: O999 and O815

   John has reservations on R221 (seat 34U), R567 (seat 23U), and R234 (seat 43L).

   Henry has reservations on O999 (seat 15L) and H008 (seat 18L).

   Duke has reservations on H897 (seat 7U) and O815 (seat 2L).
10. To what extent is the order in which SELECT and PROJECT operations are applied to a relation significant? That is, under what conditions will SELECTing and then PROJECTing produce the same results as first PROJECTing and then SELECTing?
11. Give an argument showing that the "where" clause in the JOIN operation as described in Section 9.2 is not necessary. (That is, show that any query that uses a "where" clause could be restated using a JOIN operation that concatenated every tuple in one relation with every tuple in the other.)
12. In terms of the relations shown below, what is the appearance of the relation FINAL after executing each of these instructions:

A relation

L	M	N
X	P	1
Y	Q	7
Z	R	6

B relation

X	Y
9	7
4	3

a. FINAL ← SELECT from B where X = 9
b. FINAL ← PROJECT L from A
c. FINAL ← SELECT from A where L = Z
d. FINAL ← JOIN A and B where A.N = B.Y

13. Using the commands SELECT, PROJECT, and JOIN, write a sequence of instructions to answer each of the following questions about branches and their courses in terms of the following database:

Course relation

CName	ID
Networks	IT655
DataBase	CS543
VLSI	EC653

Branch relation

BName	ID	Credits
Computer Science	CS543	4
Computer Science	EC653	5
Electronics & Communication	IT655	4
Electronics & Communication	EC653	5
Information Technology	CS543	4
Information Technology	IT655	4

a. Which branches offer IT665?
b. List all the branches present in Branch relation.
c. Which branches offer 4-credit courses?

14. Answer question 13 using SQL.

15. Using the commands `SELECT`, `PROJECT`, and `JOIN`, write sequences to answer the following questions about the information in the `EMPLOYEE`, `JOB`, and `ASSIGNMENT` relations in Figure 9.5:
a. Obtain the name of an employee whose Job ID is S25X.
b. Obtain a list of the department, skill code and job title of the employee named G. Jerry Smith.
c. Obtain a list of the name and address of the employee whose start date is 5-1-2010.

16. Answer question 14 using SQL.

17. Design a relational database containing information about authors, their titles, and their publishers. (Avoid redundancies similar to those in Figure 9.4.)

18. Design a relational database containing information about students, their courses, and their current semester along with their disciplines. (Avoid redundancies similar to those in Figure 9.4.)

19. Design a relational database containing information about employees of different companies and their salaries. (Avoid redundancies similar to those in Figure 9.4.)

20. Design a relational database containing information about fruits, colors, and the season of their availability. (Avoid redundancies similar to those in Figure 9.4.)

21. Design a relational database containing information about parts, suppliers, and customers. Each part might be supplied by several suppliers and ordered by many customers. Each supplier might supply many parts and have many customers. Each customer might order many parts from many suppliers; in fact, the same part might be ordered from more than one supplier. (Avoid redundancies similar to those in Figure 9.4.)

22. Write a sequence of instructions (using the operations `SELECT`, `PROJECT` and `JOIN`) to retrieve the `JobId`, `StartDate`, and `TermDate` for each job in the accounting department from the relational database described in Figure 9.5.

23. Answer the previous problem using SQL.

24. Write a sequence of instructions (using the operations `SELECT`, `PROJECT` and `JOIN`) to retrieve the `Name`, `Address`, `JobTitle`, and `Dept` of every current employee from the relational database described in Figure 9.5.

25. Answer the previous problem using SQL.

26. Write a sequence of instructions (using the operations `SELECT`, `PROJECT` and `JOIN`) to retrieve the `Name` and `JobTitle` of each current employee from the relational database described in Figure 9.5.

27. Answer the previous problem using SQL.

28. What is the difference in the information supplied by the single relation

Name	Department	TelephoneNumber
Jones	Sales	555-2222
Smith	Sales	555-3333
Baker	Personnel	555-4444

and the two relations

Name	Department
Jones	Sales
Smith	Sales
Baker	Personnel

Department	TelephoneNumber
Sales	555-2222
Sales	555-3333
Personnel	555-4444

29. Design a relational database containing information about novels, its authors and the number of pages in the novels. Be sure to allow for the fact that a novel may contain less number of pages in one version and at the same it may contain more number of pages in a new version.

30. Pick a popular website such as www.google.com, www.amazon.com, or www.ebay.com and design a relational database that you would propose to serve as the site's supporting database.

31. On the basis of the database represented in Figure 9.5, state the question that is answered by the following program segment:

```
TEMP ← SELECT from JOB
  where JobTitle = 'Secretary'
RESULT ← PROJECT SkillCode, Department
  from TEMP
```

32. Answer question 31 using SQL.

33. On the basis of the database represented in Figure 9.5, state the question that is answered by the following program segment:

```
INTER1 ← JOIN JOB and ASSIGNMENT
  where JOB.JobId =
    ASSIGNMENT.JobId
INTER2 ← SELECT from INTER1 where
  EmplId = '23Y34'
FINAL ← PROJECT TermDate, StartDate
  from INTER2
```

34. Answer question 33 using SQL.

35. On the basis of the database represented in Figure 9.5, state the question that is answered by the following program segment:

```
TEMP1 ← JOIN EMPLOYEE and JOB
  where EMPLOYEE.EmplId = JOB.EmplId
```

36. Translate the query in the previous problem into SQL.

37. Translate the SQL statement

```
SELECT Job.JobTitle
FROM Assignment, Job
WHERE Assignment.JobId = Job.JobId
  AND Assignment.EmplId = '34Y70';
```

into a sequence of SELECT, PROJECT, and JOIN operations.

38. Translate the SQL statement

```
SELECT Assignment.StartDate
FROM Assignment, Employee
WHERE Assignment.EmplId =
  Employee.EmplId
  AND Employee.Name = 'Joe E.
    Baker';
```

into a sequence of SELECT, PROJECT, and JOIN operations.

39. Describe the effect that the following SQL statement would have on the database in Problem 13.

```
INSERT INTO Course
VALUES ('Computer Concepts', 'CS342');
```

40. Describe the effect that the following SQL statement would have on the database in question 13.

```
UPDATE Branch
  SET ID = 'CS555'
  WHERE BName = 'Computer Science'
    AND Credits = 5;
```

***41.** Identify some of the objects that you would expect to find in an object-oriented database used to maintain a grocery store's inventory. What methods would you expect to find within each of these objects?

***42.** Identify some of the objects that you would expect to find in an object-oriented database used to maintain records of a library's holdings. What methods would you expect to find within each of these objects?

***43.** What incorrect information is generated by the following schedule of transactions T1 and T2?

T1 is designed to compute the sum of accounts A and B; T2 is designed to transfer $100 from account A to account B. T1 begins by retrieving the balance of account A; then, T2 performs its transfer; and finally, T1 retrieves the balance of account B and reports the sum of the values it has retrieved.

***44.** What is the difference between incorrect summary problem and lost update problem? Explain your answer with examples.

***45.** What effect would the wound-wait protocol have on the sequence of events in question 43 if T1 was the younger transaction? If T2 was the younger transaction?

***46.** Suppose one transaction tries to add $100 to an account whose balance is $200 while another tries to withdraw $100 from the same account. Describe an interweaving of these transactions that would lead to a final balance of $100. Describe an interweaving of these transactions that would lead to a final balance of $300.

***47.** What is the difference between a transaction having exclusive access or shared access to an item in a database and why is the distinction important?

***48.** The problems discussed in Section 9.4 involving concurrent transactions are not limited to database environments. What similar problems would arise when accessing a document with word processors? (If you have a PC with a word processor, try to access the same document with two activations of the word processor and see what happens.)

***49.** Suppose a sequential file contains 50,000 records and 5 milliseconds is required to interrogate an entry. How long should we expect to wait when retrieving a record from the middle of the file?

***50.** List the steps that are executed in the merge algorithm in Figure 9.15 if one of the input files is empty at the start.

***51.** Modify the algorithm in Figure 9.15 to handle the case in which both input files contain a record with the same key field value. Assume that these records are identical and that only one should appear in the output file.

***52.** Design a system by which a file stored on a disk can be processed as a sequential file with either of two different orderings.

***53.** Describe how a sequential file containing information about a magazine's subscribers could be constructed using a text file as the underlying structure.

***54.** Design a technique by which a sequential file whose logical records are not a consistent size could be implemented as a text file. For example, suppose you wanted to construct a sequential file in which each logical record contained information about a novelist as well as a list of that author's works.

***55.** Explain the terms cluster analysis, association analysis, outlier analysis, and sequential pattern analysis in brief.

***56.** The chapter drew parallels between a traditional file index and the file directory system maintained by an operating system. In what ways does an operating system's file directory differ from a traditional index?

***57.** If a hash file is partitioned into 10 buckets, what is the probability of at least two of three arbitrary records hashing to the same bucket? (Assume the hash function gives no bucket priority over the others.) How many records must be stored in the file until it is more likely for collisions to occur than not?

***58.** Solve the previous problem, assuming that the file is partitioned into 100 buckets instead of 10.

***59.** If we are using the division technique discussed in this chapter as a hash function and the file storage area is divided into 23 buckets, which section should we search to find the record whose key, when interpreted as a binary value, is the integer 124?

***60.** Compare the implementation of a hash file to that of a homogeneous two-dimensional array. How are the roles of the hash function and the address polynomial similar?

***61.** Give one advantage that
a. a sequential file has over an indexed file.
b. a sequential file has over a hash file.
c. an indexed file has over a sequential file.
d. an indexed file has over a hash file.
e. a hash file has over a sequential file.
f. a hash file has over an indexed file.

***62.** Explain the technique of hashing. How is it related to a hash function and a hash table?

Social Issues

The following questions are intended as a guide to the ethical/social/legal issues associated with the field of computing. The goal is not merely to answer these questions. You should also consider why you answered as you did and whether your justifications are consistent from one question to the next.

1. In the United States, DNA records of all federal prisoners are now stored in a database for use in criminal investigations. Would it be ethical to release this information for other purposes—for example, for medical research? If so, for what purposes? If not, why not? What are the pros and cons in each case?
2. To what extent should a university be allowed to release information about its students? What about their names and addresses? What about grade distributions without identifying the students? Is your answer consistent with your answer to question 1?
3. What restrictions are appropriate regarding the construction of databases about individuals? What information does a government have a right to hold regarding its citizens? What information does an insurance company have a right to hold regarding its clients? What information does a company have a right to hold regarding its employees? Should controls in these settings be implemented and, if so, how?
4. Is it proper for a credit card company to sell the purchasing patterns of its clients to marketing firms? Is it acceptable for a sports car mail order business to sell its mailing list to a sports car magazine? Is it acceptable for the Internal Revenue Service in the United States to sell the names and addresses of those taxpayers with significant capital gains to stockbrokers? If you cannot answer with an unqualified yes or no, what would you propose as an acceptable policy?
5. To what extent is the designer of a database responsible for how the information in that database is used?
6. Suppose a database mistakenly allows unapproved access to information in the database. If that information is obtained and used adversely, to what degree do the database designers share responsibility for the misuse of the information? Does your answer depend on the amount of effort required by the perpetrator to discover the flaw in the database design and obtain the unauthorized information?
7. The prevalence of data mining raises numerous issues of ethics and privacy. Is your privacy infringed if data mining reveals certain characteristics about the overall population of your community? Does the use of data mining promote good business practice or bigotry? To what extent is it proper to force citizens to participate in a census, knowing that more information will be extracted from the data than is explicitly requested by the individual questionnaires? Does data mining give marketing firms an unfair advantage over unsuspecting audiences? To what extent is profiling good or bad?
8. To what extent should a person or corporation be allowed to collect and hold information about individuals? What if the information collected is already publicly available although scattered among several sources? To what extent should a person or company be expected to protect such information?

9. Many libraries offer a reference service so that patrons can enlist the assistance of a librarian when searching for information. Will the existence of the Internet and database technology render this service obsolete? If so, would that be a step forward or backward? If not, why not? How will the existence of the Internet and database technology affect the existence of libraries themselves?

10. To what extent are you exposed to the possibility of identity theft? What steps can you take to minimize that exposure? How could you be damaged if you were the victim of identity theft? Should you be liable when identity theft occurs?

Additional Reading

Berstein, A., M. Kifer, and P. M. Lewis. *Database Systems,* 2nd ed. Boston, MA: Addison-Wesley, 2006.

Connolly, T., and C.E. Beg. *Database Systems: A Practical Approach to Design, Implementation and Management,* 5th ed. Boston, MA: Addison-Wesley, 2009.

Date, C. J. *An Introduction to Database Systems,* 8th ed. Boston, MA: Addison-Wesley, 2004.

Date, C. J. *Databases, Types and the Relational Model,* 3rd ed. Boston, MA: Addison-Wesley, 2007.

Elmasri, R., and S. Navathe. *Fundamentals of Database Systems,* 6th ed. Boston, MA: Addison-Wesley, 2011.

Patrick, J. J. *SQL Fundamentals,* 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

Silberschatz, A., H. Korth, and S. Sudarshan. *Database Systems Concepts,* 6th ed. New York: McGraw-Hill, 2009.

Ullman, J. D., and J. D. Widom. *A First Course in Database Systems,* 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2008.