Just imagine how difficult it would be to get any information from an information system if data were stored in an unorganized way, or if there was no systematic way to retrieve it. Therefore, in all information systems, data resources must be organized and structured in some logical manner so that they can be accessed easily, processed efficiently, retrieved quickly, and managed effectively. Thus, data structures and access methods ranging from simple to complex have been devised to efficiently organize and access data stored by information systems. In this section, we will explore these concepts, as well as more technical concepts of database management.

Database Structures

The relationships among the many individual records stored in databases are based on one of several logical data structures or models. Database management system packages are designed to use a specific data structure to provide end users with quick, easy access to information stored in databases. Five fundamental database structures are the hierarchical, network, relational, object-oriented, and multidimensional models. Simplified illustrations of the first three database structures are shown in Figure 7.15.

Hierarchical Structure

Early mainframe DBMS packages used the hierarchical structure, in which the relationships between records form a hierarchy or tree-like structure. In the traditional hierarchical model, all records are dependent and arranged in multilevel structures, consisting of one root record and any number of subordinate levels. Thus, all of the relationships among records are one-to-many, since each data element is related only to one element above it. The data element or record at the highest level of the hierarchy (the department data element in this illustration) is called the root element. Any data element can be accessed by moving progressively downward from a root and along the branches of the tree until the desired record (for example, the employee data element) is located.

Network Structure

The network structure can represent more complex logical relationships, and is still used by many mainframe DBMS packages. It allows many-to-many relationships among records—that is, the network model can access a data element by following one of several paths, because any data element or record can be related to any number of other data elements. For example, in Figure 7.15, departmental records can be related to more than one employee record, and employee records can be related to more than one project record. Thus, one could locate all employee records for a particular department, or all project records related to a particular employee.

Relational Structure

The relational model has become the most popular of the three database structures. It is used by most microcomputer DBMS packages, as well as many minicomputer and mainframe systems. In the relational model, all data elements within the database are viewed as being stored in the form of simple tables. Figure 7.15 illustrates the relational database model with two tables representing some of the relationships among departmental and employee records. Other tables, or relations, for this organization's database might represent the data element relationships among projects, divisions, product lines, and so on. Database management system packages
on the relational model can link data elements from various tables to provide information to users. For example, a DBMS package could retrieve and display an employee’s name and salary from the employee table in Figure 7.15, and the name of his or her department from the department table, by using their common department number field (Deptno) to link or join the two tables.
The checking and savings account objects can inherit common attributes and operations from the bank account object.

**Object-Oriented Structure**

Other database models are being developed to provide capabilities missing from the hierarchical, network, and relational structures. One example is the object-oriented database model. We introduced the concept of objects when we discussed object-oriented programming in Chapter 5. As Figure 7.16 illustrates, an object consists of data values describing the attributes of an entity, plus the operations that can be performed upon the data. This capability is called encapsulation, and it allows the object-oriented model to better handle more complex types of data (graphics, pictures, voice, text) than other database structures. The object-oriented model also supports inheritance; that is, new objects can be automatically created by replicating some or all of the characteristics of one or more parent objects. Thus, in Figure 7.16, the checking and savings account objects can both inherit the common attributes and operations of the parent bank account object. Such capabilities have made object-oriented database management systems (OODBMS) popular in computer-aided design (CAD) and similar applications. For example, they allow designers to develop product designs, store them as objects in an object-oriented database, and replicate and modify them to create new product designs.

The multidimensional database model uses multidimensional structures to store data and relationships between data. You can visualize multidimensional structures as cubes of data—and cubes within cubes of data. Each side of the cube is considered a dimension of the data. Figure 7.17 is an example that shows that each dimension can represent a different category, such as product type, region, sales channel, and time. Each cell within the multidimensional structure contains aggregated data related to elements along each of the dimensions. For example, a single cell may contain the total sales for a product in a region for a specific sales channel in a single
FIGURE 7.17
An example of the different dimensions of a multidimensional database.

![Diagram of a multidimensional database]


A major benefit of multidimensional databases is that they are a compact and easy-to-understand way to visualize and manipulate data elements that have many interrelationships. So multidimensional databases have become the most popular database structure for the analytical databases that support online analytical processing (OLAP) applications, in which fast answers to complex business queries are expected. We will discuss OLAP applications in Chapter 10.

The hierarchical data structure is a natural model for the databases used for many of the structured, routine types of transaction processing characteristic of many business operations. Data for many of these operations can easily be represented by groups of records in a hierarchical relationship. However, there are many cases where information is needed about records that do not have hierarchical relationships. For example, it is obvious that, in some organizations, employees from more than one department can work on more than one project (see Figure 7.15). A
Database Development

FIGURE 7.18
Creating a database with Microsoft Access. This display shows how a customer record is created and added to the Customers table that is part of a company database.

Developing small, personal databases is relatively easy using microcomputer database management packages. See Figure 7.18. However, developing a large database can be a complex task. In many companies, developing and managing large corporate databases is the primary responsibility of the database administrator and database design analysts. They work with end users and systems analysts to deter-
mine (1) what data definitions should be included in the database and (2) what structure or relationships should exist among the data elements.

As Figure 7.19 illustrates, database development may start with a top-down data planning process. Database administrators and designers work with corporate and end user management to develop an enterprise model that defines the basic business processes of the enterprise. Then they define the information needs of end users in a business process, such as the purchasing/receiving process that all businesses have [16].

Next, end users must identify the key data elements that are needed to perform their specific business activities. This frequently involves developing entity relationship diagrams (ERDs) that model the relationships among the many entities involved in business processes. For example, Figure 7.20 illustrates some of the relationships in a purchasing/receiving system. End users and database designers could use ERD models to identify what supplier and product data are necessary in the purchasing/receiving process. These user views are a major part of a data modeling process.

**Data Planning and Database Design**

**FIGURE 7.19**
Database development involves data planning and database design activities. Data models that support business processes are used to develop databases that meet the information needs of users.
FIGURE 7.20
This entity relationship diagram illustrates some of the relationships among entities in a purchasing/receiving system.

process where the relationships between data elements are identified. Each data model defines the logical relationships among the data elements needed to support a basic business process. For example, can a supplier provide more than one type of product to us? Can a customer have more than one type of account with us? Can an employee have several pay rates or be assigned to several project work groups? Answering such questions will identify data relationships that have to be represented in a data model that supports a business process.

These data models then serve as logical frameworks (called schemas and sub-schemas) on which to base the physical design of databases and the development of application programs to support the business processes of the organization. A schema is an overall logical view of the relationships between data in a database, while the subschema is a logical view of the data relationships needed to support specific end user application programs that will access that database.

Remember that data models represent logical views of the data and relationships of the database. Physical database design takes a physical view of the data (also called the internal view) which describes how data is to be physically arranged, stored, and accessed on the magnetic disks and other secondary storage devices of a computer system. For example, Figure 7.21 shows these different database views and the software interface of a bank database processing system.

Accessing Files and Databases

Databases and files are stored on various types of storage media and are organized in a variety of ways to make it easier to access the data records they contain. In database and file maintenance, records have to be continually added, deleted, or updated to reflect business transactions. Data must also be accessed so information can be produced in response to end user requests. Thus, efficient access to data is important.

Key Fields

That's why all data records usually contain one or more identification fields, or keys, that identify the record so it can be located. For example, the social security number of a person is often used as a primary key field that uniquely identifies the data records of individuals in student, employee, and customer files and databases. Other methods can be used to identify and link data records stored in several different database files. For example, hierarchical and network databases may use pointer fields. These are fields within a record that indicate (point to) the location of another record that is related to it in the same file, or in another file. Hierarchical and net-
FIGURE 7.21
Examples of the logical and physical database views and the software interface of a database processing system in banking.

Logical User Views
Data elements and relationships (the subschemas) needed for checking, savings, or installment loan processing

Data elements and relationships (the schema) needed for the support of all bank services

Software Interface
The DBMS provides access to the bank’s databases

Physical Data Views
Organization and location of data on the storage media

Relational database management packages use primary keys to link records. Each table (file) in a relational database must contain a primary key. This field (or fields) uniquely identifies each record in a file and must also be found in other related files. For example, in Figure 7.15, department number (Deptno) is the primary key in the Department table and is also a field in the Employee table. As we mentioned earlier, a relational database management package could easily provide you with information from both tables to join the tables and retrieve the information you want. See Figure 7.22.

One of the basic ways to access data is to use a sequential organization, in which records are physically stored in a specified order according to a key field in each record. For example, payroll records could be placed in a payroll file in a numerical order based on employee social security numbers. Sequential access is fast and efficient when dealing with large volumes of data that need to be processed periodically. However, it requires that all new transactions be sorted into the proper sequence for sequential access processing. Also, most of the database or file may have to be searched, store, or modify even a small number of data records. Thus, this method is too slow to handle applications requiring immediate updating or responses.
When using **direct access** methods, records do not have to be arranged in any particular sequence on storage media. However, the computer must keep track of the storage location of each record using a variety of **direct organization** methods so that data can be retrieved when needed. New transactions data do not have to be sorted, and processing that requires immediate responses or updating is easily handled. There are a number of ways to directly access records in the direct organization method. Let's take a look at three widely used methods to accomplish such direct access processing.

One common technique of direct access is called **key transformation**. This method performs an arithmetic computation on a key field of record (such as a product number or social security number) and uses the number that results from that calculation as an address to store and access that record. Thus, the process is called **key transformation** because an arithmetic operation is applied to a key field to transform it into the storage location address of a record. Another direct access method used to store and locate records involves the use of an **index** of record keys and related storage addresses. A new data record is stored at the next available location, and its key and address are placed in an index. The computer uses this index whenever it must access a record.

In the **indexed sequential access method** (ISAM), records are physically stored in a sequential order on a magnetic disk or other direct access storage device based on the key field of each record. In addition, each file contains an index that references one or more key fields of each data record to its storage location address. Thus, an individual record can be directly located by using its key fields to search and locate its address in the file index, just as you can locate key topics in this book by looking them up in its index. As a result, if a few records must be processed quickly, the file index is used to directly access the record needed. However, when large numbers of records must be processed periodically, the sequential organization provided by this method is used. For example, processing the weekly payroll for employees or producing monthly statements for customers would be done using sequential access processing of the records in the file or database.