Contents

Chapter

1

Basic Concepts

An organization must have accurate and reliable data for effective decision making. To this end, the organization maintains records on the various facets of its operations by building appropriate models of the diverse classes of objects of interest. These models capture the essential properties of the objects and record relationships among them. Such related data is called a **database.** A **database system** is an integrated collection of related files, along with details of the interpretation of the data contained therein.

A **database management system** (DBMS) is a software system that allows access to data contained in a database. The objective of the DBMS is to provide a convenient and effective method of defining, storing, and retrieving the information contained in the database. The DBMS interfaces with application programs, so that the data contained in the database can be used by multiple applications and users. In this chapter we look at the structure of a database management system, its main components and their interactions, and the different classes of users. The database system allows these users to access and manipulate the data contained in the database in a convenient and effective manner. In addition the DBMS exerts centralized control of the database, prevents fraudulent or unauthorized users from accessing the data, and ensures the privacy of the data.

1.1 Data Modeling for a Database

An organization is established to undertake one or several operations or projects. Typically, it is an environment with a single administrative control. Examples of an organization are a bank, conglomerate, government, hospital, manufacturer, or university. An organization may be a single venture such as a university located on a single campus under a single board of governors, or it may consist of a number of units, each of which could be considered a separate organization. An instance of the latter is a conglomerate, which is made up of various quasi-independent enterprises.

All organizations have some basic, common functions. Typically an organization needs to collect, process, store, and disseminate data for its human, financial, and material resources and functions. The functions performed by an organization depend on its nature and purpose and could include some of the following: payroll, accounts receivable and payable, sales reports and forecasts, design and manufacturing, course offerings, course enrollment, student transcripts, medical histories. The database system is an attempt to consolidate under a single administration the collection, storage, and dissemination of the data required for these operations.

The database is used to store information useful to an organization. To represent this information, some means of modeling is used. The components used in modeling are limited to the objects of interest to the organization and the relationships among these objects. One category of objects of concern to any organization is its personnel, and one relationship that exists within this category of objects is that of supervisor to employees. Another area in which the definition, management, and manipulation of a considerable amount of data is required is in computer-aided design (CAD) and computer-aided manufacturing (CAM). The objects in these applications consist of the specifications of various components and their interrelationships.

Each category of objects has certain characteristics or properties, called its attributes. Relationships have certain properties as well, represented as the attributes of

the relationship. We briefly look at these components of modeling in this chapter and defer detailed discussion of data modeling to the next chapter.

1.1.1 Entities and Their Attributes

Entities are the basic units used in modeling classes of concrete or abstract objects. Entities can have concrete existence or constitute ideas or concepts. Each of the following is an entity: building, room, chair, transaction, course, machine, employee. An **entity type** or **entity set** is a group of similar objects of concern to an organization for which it maintains data. Examples of entity sets are transactions, concepts, job positions, courses, employees, inventories of raw and finished products, inventories of plants and machinery, students, academic staff, nonacademic staff, managers, flight crews, flights and reservations.

Identifying and classifying objects into entity sets can be difficult, because an object can belong to different entity sets simultaneously. A person can be a student as well as a part-time employee. Consider the modeling of a flight crew. It consists of a group of individuals employed by an organization who belong to the entity sets EMPLOYEE and PERSON. These individual members of the flight crew have different skills and functions. Some are assigned to the flight deck, others make up the cabin crew. In modeling we may decide simply to use the entity set EMPLOYEE and add the attribute *Skill* with possible values such as pilot, first officer, navigator, engineer, steward, purser, and stewardess. A *FLIGHT_CREW* can then be considered as a **relationship** among the instances of the entity set EMPLOYEE with appropriate value of *Skill*. Or we could consider creating entity sets PILOT, FLIGHT_ENGINEER, NAVIGATOR, and so forth for each distinct group of employees required in a flight crew. We can then set up a relationship, let us call it *FLIGHT_CREW*, among these entity sets.

One of the first steps in data modeling is to identify and select the entity sets that will best organize useful information for the database application (see Figure 1.1). Problems to be resolved include delimiting an entity and distinguishing and identifying occurrences of entities of the same type. In effect, entities such as bolts, electrons, trees, or cattle cannot be uniquely identified. However, with these types of entities, their number, density, weight, or other such attributes may be sufficient for modeling. For instance, we want to distinguish a #8-24 bolt that is two inches long from a #10-24 bolt of the same length. However, an instance of the former need not be distinguished from another instance of the same. Another problem to be resolved is the method of handling the changes that occur in an entity over time. An instance of the entity EMPLOYEE could successively be a junior engineer, an engineer, a senior engineer, and a manager.

To store data on an entity set, we have to create a model for it. For example, employees of an organization are modeled by the entity set EMPLOYEE. We must include in the model the properties or characteristics of employees that may be useful to the organization. Some of these properties are EMPLOYEE.*Name*, EMPLOYEE.*Soc_Sec_No* (Social Security number), EMPLOYEE.*Address*, EMPLOYEE.*Skill*, EMPLOYEE.*Annual_Salary*. Other properties, which are not deemed useful to the organization and not recorded, could be the color of the employees' hair or the size of the shoes they wear. The properties that characterize an entity set are called its

EE. *Annual_Salary* and EMPLOYEE.*Soc_Sec_No* would have a domain consisting of positive nine-digit integers. Although the set of values for the two attributes are identical, their domains are treated differently because we interpret the salary as a monetary unit and the Social Security number as an identifying number.

Keys

A **key** is a single attribute or combination of two or more attributes of an entity set that is used to identify one or more instances of the set. The attribute EMPLOYEE.*Soc_Sec_No* uniquely identifies an instance of the entity set EMPLOYEE. The value 787394510 for the attribute EMPLOYEE.*Soc_Sec_No* uniquely identifies the employee George Hall. A key would not be unique if an attribute such as EMPLOYEE.*Skill* were used. Such attributes identify more than one instance of the entity set EMPLOYEE. The value of cook for EMPLOYEE.*Skill* identifies all employees with this skill.

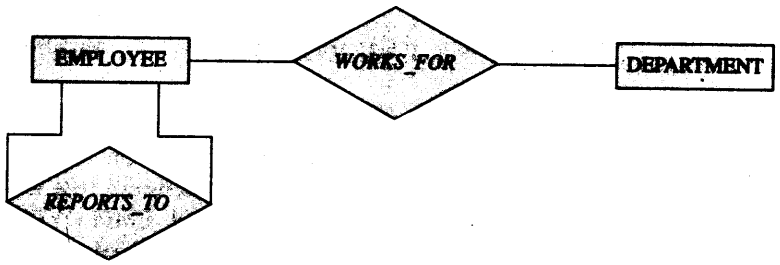
Two instances of an entity set could have the same values for all its attributes. In the case of the entity set GUEST, it is likely that the two guests Don Smith and David Smith, who are identical twins living at 123 New Brunswick Drive, are both registered as D. Smith. To distinguish such instances, we introduced the attribute GUEST.*Soc_Sec_No*. This attribute is unique and will identify an instance of the entity set GUEST. Such a unique entity identifier as GUEST.*Soc_Sec_No* is referred to as a **primary key.**

If we add additional attributes to a primary key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called **superkeys:** a primary key is, therefore, a minimum superkey. It is possible that some existing attribute or combination of attributes of an entity set uniquely identifies an instance of the set. In this case, additional attributes need not be introduced. However, if no such attribute or combination of attributes exists, then in order to identify the object uniquely, an additional attribute needs to be introduced. Examples of such additional attributes are found in the introduction of identifiers such as car serial numbers, part numbers, customer and account numbers to uniquely identify cars, parts, customers and accounts, respectively. Instances of these entities would be harder to distinguish by their other attributes. Suppose that George Hall banks with the First National Bank. Even though each customer has a unique *Soc_Sec_No,* the bank uses a unique identifier called the *Account_Number* to identify each account. The fact that George Hall may have more than one account of the same type, for example, two current accounts, three savings accounts, and a mortgage account, necessitates such identification. The attribute *Account_Number* is a better choice for the primary key of the entity set ACCOUNT than the attribute *Soc_Sec_No*.

There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called **candidate keys.** In such a case we must decide which of the candidate keys will be used as the primary key. The remaining candidate keys would be considered **alternate keys.**

A **secondary key** is an attribute or combination of attributes that may not be a candidate key but that classifies the entity set on a particular characteristic. A case in point is the entity set EMPLOYEE having the attribute *Department,* which identifies by its value all instances of EMPLOYFE who belong to a given department. More

Figure 1.4 Relationships between entity sets.



than one employee may belong to a department, so the *Department* attribute is not a candidate key for the entity set EMPLOYEE, since it cannot uniquely identify an individual employee. However, the *Department* attribute does identify all employees belonging to a given department.

1.1.2 Relationships

The **relationship set** is used in data modeling to represent an association between entity sets. This association could have certain properties represented by the attributes of the relationship set. A *Grade* is an attribute of the ENROLLMENT relationship set between the entity sets COURSE and STUDENT.

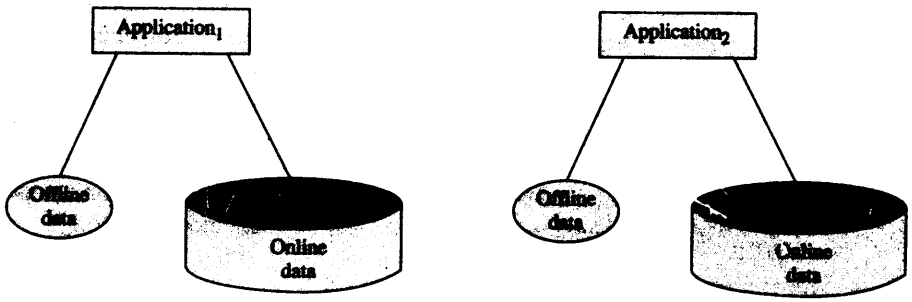
Each relationship set is named. The fact that an employee is assigned to a given department is indicated by the named relationship set *WORKS_FOR* between the entity sets EMPLOYEE and DEPARTMENT. Compare this with using the attribute *Department* as an attribute of EMPLOYEE. Figure 1.4 shows this relationship set as a diamond connected to the entity sets involved in the relationship. There could be a number of entity sets involved in a relationship and the same entity set could be involved in a number of different relationship sets. The relationship set *REPORTS_TO* in Figure 1.4 involves the same entity set EMPLOYEE and indicates that an employee reports to another employee, the supervisor. The same entity set EMPLOYEE is involved in both these relationship sets. We discuss the concept of relationships further in the next chapter.

1.2 Records and Files

The physical representation of an entity set is made by aggregating the attributes used to model the entity set. Such a representation is called a **record type.** An instance of a record type is a **record occurrence.** The usual practice is to group together in predetermined order the values of the attributes of an instance of an entity set and store them in an appropriate storage medium. Therefore,

[George Hall, 787394510, 110 Woolsey Drive, cook, 42650.00]

Figure 1.7 Nondatabase environment without any shared data.



Consider two applications that require data on the entity set EMPLOYEE. The first application involves the public relations department sending each employee a newsletter and related material. This material is mailed to employees' homes, necessitating printing mailing labels. The application, therefore, is interested in the record type EMPLOYEE, which contains the values for the attributes EMPLOYEE.*Name* and EMPLOYEE.*Address*. This record type is the view of the real world as far as this application is concerned and can be described in pseudocode as shown in Figure 1.8.

The second application involves the payroll application for paycheck preparation. It requires the record type EMPLOYEE, which contains the values for the attributes EMPLOYEE.*Name*, EMPLOYEE.*Soc_Sec_No*, *EMPLOYEE.Address*, and EMPLOYEE.*Annual_Salary*. This record type is shown in Figure 1.9.

In a nondatabase environment, each application program is responsible for maintaining the currency of the data and a change in a data item must be effected in each copy of the data. Therefore, if an employee changes her or his address, each application program using the EMPLOYEE entity set with the attribute EMPLOYEE.*Address* would be required to update the address of that employee.

As shown in Figure 1.10, in a database environment data can be shared by these two applications. Their requirements can be integrated by the person (or a group of persons) who has the responsibility of centralized control. Such a person is referred to as the database administrator or DBA. The integrated version could appear as a record containing the following attributes: EMPLOYEE.*Name*, EMPLOYEE.*Soc_Sec_No*, EMPLOYEE.*Address*, *EMPLOYEE.Skill*, and EMPLOYEE.*Annual_Salary*. This integrated record type is shown in Figure 1.11. Note the inclusion of the

Figure 1.8 The view for the public relations application.

```
type EMPLOYEE = record
     EMPLOYEE.Name: string;
     EMPLOYEE.Address: string;
     end
```

Figure 1.9 The view for the payroll application.

```
type EMPLOYEE = record
    EMPLOYEE.Name: string;
    EMPLOYEE.Soc_Sec_No: integer;
    EMPLOYEE.Address: string;
    EMPLOYEE.Annual_Salary: integer;
    end
```

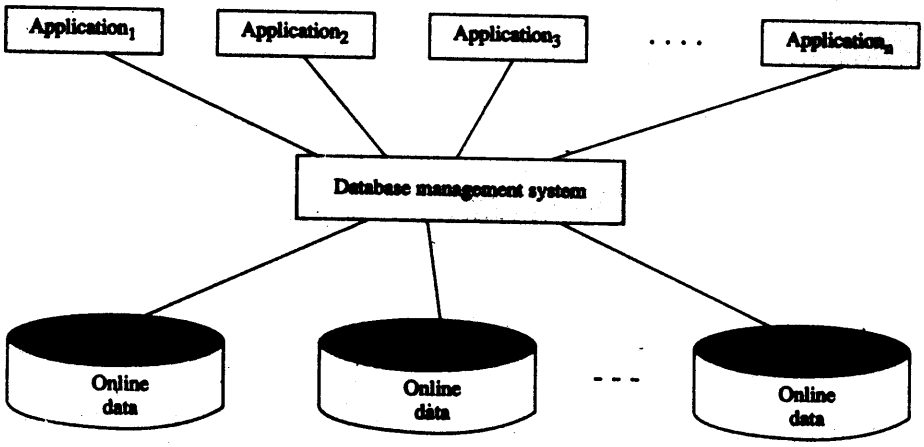
attribute EMPLOYEE.*Skill*, which is not being used by either of the above described applications.

The integrated record EMPLOYEE described above can be considered a **conceptual record.** The views of the two applications it supports can be derived from it by using appropriate **mapping,** which in this case is done by simply hiding (i.e., masking out) the unnecessary attributes. The two views of this record as seen by the two applications are shown in Figure 1.12. Each application views only a portion of the conceptual record. The record each application is concerned with is called a **logical record.**

In addition to masking out the irrelevant attributes, it is possible to have a view that contains one or more attributes obtained by computation from the conceptual record. For instance, a new application that requires the monthly salary for each employee can be supported by the conceptual record of Figure 1.11. The monthly salary is derived by a simple computation on the data in the database for the attribute EMPLOYEE.*Annual_Salary*.

The application programs discussed above can continue to view the employee record in the same manner as before; however, they no longer are required to contain information about the file structure. Any change in the storage structure, storage

Figure 1.10 Database environment with shared data.



lowest level, a description of the actual method of storing the data, is the **internal view.** The database system can be designed using these levels of abstractions as described in the following section.

1.4 The Three-Level Architecture Proposal for a DBMS

In this section we describe the generalized architecture of a database system called the ANSI/SPARC[1] model. A large number of commercial systems and research database models fit this framework. The architecture, shown in Figure 1.14, is divided into three levels: the **external level,** the **conceptual level,** and the **internal level.**

The view at each of these levels is described by a **scheme.** A scheme is an outline or a plan that describes the records and relationships existing in the view. The word scheme, which means a systematic plan for attaining some goal, is used interchangeably in the database literature with the word **schema.** The word schemas is used in the database literature for the plural instead of schemata, the grammatically correct word. The scheme also describes the way in which entities at one level of abstraction can be mapped to the next level.

External or User View

The external or user view is at the highest level of database abstraction where only those portions of the database of concern to a user or application program are included. Any number of user views (some of which may be identical) may exist for a given global or conceptual view.

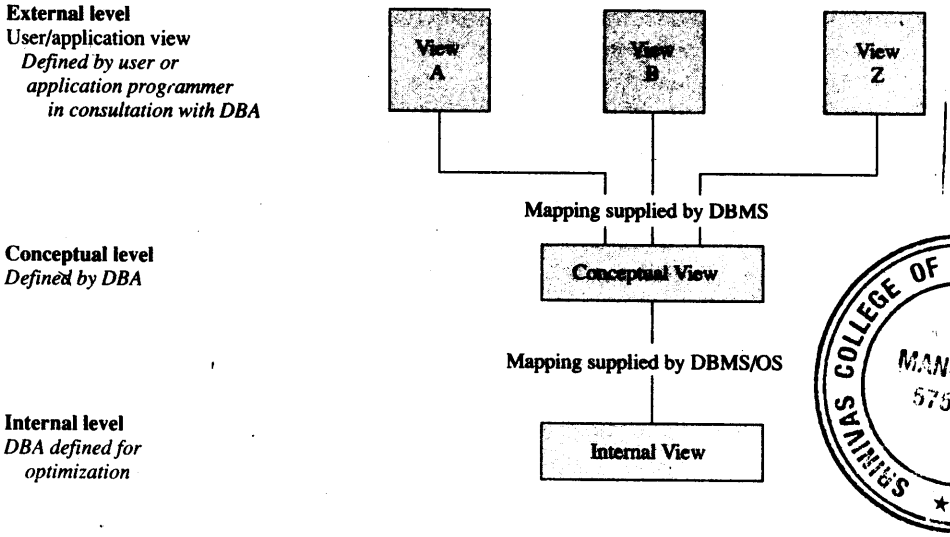
Each external view is described by means of a scheme called an **external schema.** The external schema consists of the definition of the logical records and the relationships in the external view. The external schema also contains the method of deriving the objects in the external view from the objects in the conceptual view. The objects includes entities, attributes, and relationships. (The terms view, scheme, and schema are sometimes used interchangeably when there is no confusion as to what is implied.)

Conceptual or Global View

At this level of database abstraction all the database entities and the relationships among them are included. One conceptual view represents the entire database. This conceptual view is defined by the **conceptual schema.** It describes all the records and relationships included in the conceptual view and, therefore, in the database. There is only one conceptual schema per database. This schema also contains the

[1]ANSI/SPARC: American National Standards Institute/Standards Planning and Requirements Committee

Figure 1.14 The three levels of architecture of a DBMS.



method of deriving the objects in the conceptual view from the objects in the internal view.

The description of data at this level is in a format independent of its physical representation. It also includes features that specify the checks to retain data consistency and integrity.

Internal View

We find this view at the lowest level of abstraction, closest to the physical storage method used. It indicates how the data will be stored and describes the data structures and access methods to be used by the database. The internal view is expressed by the **internal schema**, which contains the definition of the stored record, the method of representing the data fields, and the access aids used.

1.4.1 Mapping between Views

The **conceptual database** is the model or abstraction of the objects of concern to the database. Thus, the conceptual record of Figure 1.13 is the conceptual database and represents the abstraction of all the applications involving the entity set EMPLOYEE, for the present discussions. The view is the subset of the objects modeled in the conceptual database that is used by an application. There could be any number of views of a conceptual database. A view can be used to limit the portion of the database that is known and accessible to a given application.

Two mappings are required in a database system with three different views as shown in Figure 1.14. A mapping between the external and conceptual views gives the correspondence among the records and the relationships of the external and con-

Figure 1.15 External schemes of (a) user in public relations department and (b) user in payroll department.

```
type EMPLOYEE = record
    EMPLOYEE.Name: string;
    EMPLOYEE.Address: string;
    end
```

(a)

```
type EMPLOYEE = record
    EMPLOYEE.Name: string;
    EMPLOYEE.Soc_Sec_No: integer unique;
    EMPLOYEE.Address: string;
    EMPLOYEE.Salary: integer;
    end
```

(b)

public relations department given in Figure 1.15a from the conceptual view given in Figure 1.16 is to map the first and fourth fields of the record EMPLOYEE in the conceptual scheme into the first and second field of the record EMPLOYEE of the external scheme.

Figure 1.17 presents the internal level definition corresponding to the conceptual record type defined in Figure 1.16. The scheme indicates that the record EMPLOYEE is a record of length 120 bytes. There are six fields in this record and the scheme gives their sizes, types, and relative position from the beginning of the record. It also indicates that for faster access in random order, an index is to be built using the values from the primary key field EMPLOYEE.*Soc_Sec_No*.

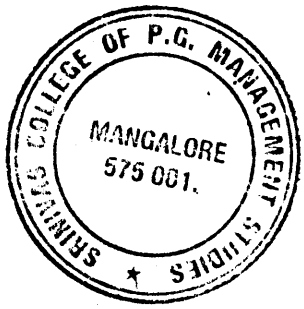
Consider a change in the conceptual view such as merging two records into one or adding fields to an existing record. This would require a change in the mapping (for external views that are based on the records undergoing changes) from the external view to the conceptual view so as to leave the external view unchanged. However, not all changes in the conceptual schema can be absorbed by the adjustment of the mapping. Some changes, such as the deletion of a conceptual view field or rec-

Figure 1.16 Conceptual schema portion of database corresponding to Figure 1.15.

```
type EMPLOYEE = record
    EMPLOYEE.Name: string;
    EMPLOYEE.Soc_Sec_No: integer primary key;
    EMPLOYEE.Department: string;
    EMPLOYEE.Address: string;
    EMPLOYEE.Skill: string;
    EMPLOYEE.Annual_Salary: integer;
    end
```

Figure 1.17 Internal schema of the portion of database corresponding to Figure 1.16.

```
type EMPLOYEE = record length 120
    EMPLOYEE.Name: string length 25 offset 0;
    EMPLOYEE.Soc_Sec_No: integer positive
                         9 dec digits offset 25
                         unique
                         use for index;
    EMPLOYEE.Department: string length 6 offset 34;
    EMPLOYEE.Address: string length 51 offset 40;
    EMPLOYEE.Skill: string length 20 offset 91;
    EMPLOYEE.Salary: integer positive 9,2 dec
                         digits offset 111;
    end
```



ord, may require changes in the external view and application programs using this external view.

Physical data independence is achieved by the presence of the internal level of the database and the mapping or transformation from the conceptual level of the database to the internal level. Conceptual level to internal level mapping, therefore, provides a means to go from the conceptual view (conceptual records) to the internal view and thence to the stored data in the database (physical records). If there is a need to change the file organization or the type of physical device used as a result of growth in the database or new technology, a change is required in the transformation functions between the physical and conceptual levels. This change is necessary to maintain the conceptual level invariant. Altering the physical database organization, however, can affect the response and efficiency of existing application programs. This may mean that while some application programs run faster, others may be slowed down. Regardless, no changes are required in the application programs themselves and they will run correctly with the new physical data organization.

The physical data independence criterion requires that the conceptual level does not specify storage structures or the access methods (indexing, hashing method, etc.) used to retrieve the data from the physical storage medium. Making the conceptual schema physically dataindependent means that the external schema, which is defined on the conceptual schema, is in turn physically dataindependent.

Another aspect of data independence allows different interpretations of the same data. The storage of data is in bits and may change from EBCDIC to ASCII coding, SI (metric) to imperial units of measure, or the data may be compressed to save storage space without affecting the application programs. In addition, a data field required by an application may be derived from one or several fields from one or more records of the database. As mentioned earlier, a field such as EMPLOYEE.*Age* may be derived from the stored field EMPLOYEE.*Birthdate* and from the calendar function DATE usually provided by the operating system. This is an example of a **virtual field.** Another such virtual field could be *Total_Hours_Worked_For_Week*, which is derived from the total of the seven entries for *Hours_Worked_During_Week* (record of hours worked on each day of the week). Note that unlike a real field, a virtual field may not be directly modified by a user.

1.5 Components of a DBMS

Let us now examine the components and structure of a database management system. A DBMS is a complex software system that is used to manage, store, and manipulate data and the metadata used to describe the data. It is utilized by a large variety of users, from the very naive to the most sophisticated, to retrieve and manipulate data under its control. The users could be utilizing the database concurrently from online terminals and/or in a batch environment via application programs written in a high-level language. Before looking at the various components of the DBMS, let us classify its users and examine the facilities it provides for the definition and manipulation of data.

1.5.1 Classification of DBMS Users

The users of a database system can be classified in the following groups, depending on their degree of expertise or the mode of their interactions with the DBMS.

Naive Users

Users who need not be aware of the presence of the database system or any other system supporting their usage are considered naive users. A user of an automatic teller machine falls in this category. The user is instructed through each step of a transaction; he or she responds by pressing a coded key or entering a numeric value. The operations that can be performed by this class of users are very limited and affect a precise portion of the database; in the case of the user of the automatic teller machine, only one or more of her or his own accounts. Other such naive users are end users of the database who work through a menu-oriented application program where the type and range of response is always indicated to the user. Thus, a very competent database designer could be allowed to use a particular database system only as a naive user.

Online Users

These are users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program. These users are aware of the presence of the database system and may have acquired a certain amount of expertise in the limited interaction they are permitted with the database through the intermediary of the application program. The more sophisticated of these users may also use a data manipulation language to manipulate the database directly. Online users can also be naive users requiring additional help, such as menus.

Application Programmers

Professional programmers who are responsible for developing application programs or user interfaces utilized by the naive and online users fall into this category. The application programs could be written in a general-purpose programming language such as Assembler, C, COBOL, FORTRAN, Pascal, or PL/I and include the commands required to manipulate the database.

Database Administrator

Centralized control of the database is exerted by a person or group of persons under the supervision of a high-level administrator. This person or group is referred to as the **database administrator (DBA).** They are the users who are most familiar with the database and are responsible for creating, modifying, and maintaining its three levels.

The DBA is the custodian of the data and controls the database structure. The DBA administers the three levels of the database and, in consultation with the overall user community, sets up the definition of the global view or conceptual level of the database. The DBA further specifies the external view of the various users and applications and is responsible for the definition and implementation of the internal level, including the storage structure and access methods to be used for the optimum performance of the DBMS. Changes to any of the three levels necessitated by changes or growth in the organization and/or emerging technology are under the control of the DBA. Mappings between the internal and the conceptual levels, as well as between the conceptual and external levels, are also defined by the DBA. Ensuring that appropriate measures are in place to maintain the integrity of the database and that the database is not accessible to unauthorized users is another responsibility. The DBA is responsible for granting permission to the users of the database and stores the profile of each user in the database. This profile describes the permissible activities of a user on that portion of the database accessible to the user via one or more user views. The user profile can be used by the database system to verify that a particular user can perform a given operation on the database.

The DBA is also responsible for defining procedures to recover the database from failures due to human, natural, or hardware causes with minimal loss of data. This recovery procedure should enable the organization to continue to function and the intact portion of the database should continue to be available.

1.5.2 DBMS Facilities

Two main types of facilities are provided by a DBMS:

- The data definition facility or data definition language (DDL).
- The data manipulation facility or data manipulation language (DML).

Data Definition Language

Database management systems provide a facility known as **data definition language (DDL)**, which can be used to define the conceptual scheme and also give some details about how to implement this scheme in the physical devices used to store the data. This definition includes all the entity sets and their associated attributes as well as the relationships among the entity sets. The definition also includes any constraints that have to be maintained, including the constraints on the value that can be assigned to a given attribute and the constraints on the values assigned to different attributes in the same or different records. These definitions, which can be described as metadata about the data in the database, are expressed in the DDL of the DBMS and maintained in a compiled form (usually as a set of tables). The compiled form of the definitions is known as a **data dictionary, directory,** or **system catalog.** The data dictionary contains information on the data stored in the database and is consulted by the DBMS before any data manipulation operation.

The database management system maintains the information on the file structure, the method used to efficiently access the relevant data (i.e., the access method). It also provides a method whereby the application programs indicate their data requirements. The application program could use a subset of the conceptual data definition language or a separate language. The database system also contains mapping functions that allow it to interpret the stored data for the application program. (Thus, the stored data is transformed into a form compatible with the application program.)

The internal schema is specified in a somewhat similar data definition language called data storage definition language. The definition of the internal view is compiled and maintained by the DBMS. The compiled internal schema specifies the implementation details of the internal database, including the access methods employed. This information is handled by the DBMS; the user need not be aware of these details.

Data Manipulation Language

The language used to manipulate data in the database is called **data manipulation language (DML).** Data manipulation involves retrieval of data from the database, insertion of new data into the database, and deletion or modification of existing data. The first of these data manipulation operations is called a **query.** A query is a statement in the DML that requests the retrieval of data from the database. The subset of the DML used to pose a query is known as a **query language;** however, we use the terms DML and query language synonymously.

The DML provides commands to select and retrieve data from the database. Commands are also provided to insert, update, and delete records. They could be used in an interactive mode or embedded in conventional programming languages such as Assembler, COBOL, FORTRAN, Pascal, or PL/I. The data manipulation functions provided by the DBMS can be invoked in application programs directly by procedure calls or by preprocessor statements. The latter would be replaced by appropriate procedure calls by either a preprocessor or the compiler. An example of a procedure call and a preprocessor statement is given below:

Procedure call: Call Retrieve (EMPLOYEE.*Name*, EMPLOYEE.*Address*)
Preprocessor statement: %select EMPLOYEE.*Name*, EMPLOYEE.*Address*
from EMPLOYEE;

These preprocessor statements, indicated by the presence of the leading % symbol, would be replaced by data manipulation language statements in the compiled version of the application program. Commands in the conventional languages allow permissible operations on the database such as data retrieval, addition, modification, or deletion.

The DML can be procedural; the user indicates not only what to retrieve but how to go about retrieving it. If the DML is nonprocedural, the user has to indicate only what is to be retrieved. The DBMS in this case tries to optimize the exact order of retrieving the various components to make up the required response.

Data definition of the external view in most current DBMSs is done outside the application program or interactive session. Data manipulation is done by procedure calls to subroutines provided by a DBMS or via preprocessor statements. In an integrated environment, data definition and manipulation are achieved using a uniform set of constructs that forms part of the user's programming environment.

1.5.3 Structure of a DBMS

For our purposes, we may assume that the database management system is structured and interfaces with various users as shown in Figure 1.18. The major components of this system are described below.

Data Definition Language Compiler

The DDL **compiler** converts the data definition statements into a set of tables. These tables contain the metadata concerning the database and are in a form that can be used by other components of the DBMS.

Data Manager

The **data manager** is the central software component of the DBMS. It is sometimes referred to as the database control system. One of the functions of the data manager is to convert operations in the user's queries coming directly via the query processor or indirectly via an application program from the user's logical view to a physical file system. The data manager is responsible for interfacing with the file system. In addition, the tasks of enforcing constraints to maintain the consistency and integrity of the data, as well as its security, are also performed by the data manager. Synchronizing the simultaneous operations performed by concurrent users is under the control of the data manager. It is also entrusted with backup and recovery operations. We discuss backup and recovery, concurrency control, and security and integrity in Chapters 11, 12, and 13, respectively.

version of the original data manipulation statements. These data manipulation operations are executed by the data manager. The data manager transfers data to or from a work area indicated in the subroutine call and control returns to the application program.

For online users who manipulate the database through the intermediary of a user interface (such as a form-based or menu-driven system) and a supporting application program written in a high-level language, the interaction is indirect. A user action that requires a database operation causes the application program to request the service via its run-time system and the data manager.

Batch users of the database also interact with the database via their application program, its run-time system, and the data manager.

Telecommunication System

Online users of a computer system, whether remote or local, communicate with it by sending and receiving messages over communication lines. These messages are routed via an independent software system called a telecommunication system or a communication control program. Examples of these programs are CICS, IDMS-DC, TALKMASTER, and IERCOMM. The telecommunication system is not part of the DBMS but the DBMS works closely with the system; the subject is covered extensively in (Cyps 78). The online user may communicate with the database directly or indirectly via a user interface (menudriven or formbased) and an application program. Messages from the user are routed by the telecommunication system to the appropriate target and responses are sent back to the user.

Data Files

Data files contain the data portion of the database.

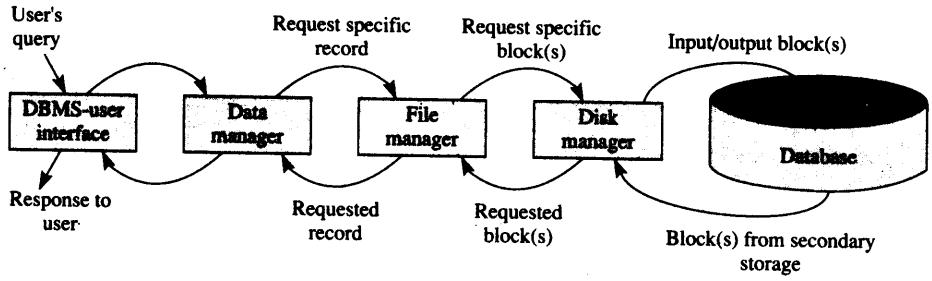
Data Dictionary

Information pertaining to the structure and usage of data contained in the database, the metadata, is maintained in a **data dictionary.** The term **system catalog** also describes this metadata. The data dictionary, which is a database itself, documents the data. Each database user can consult the data dictionary to learn what each piece of data and the various synonyms of the data fields mean.

In an integrated system (i.e., in a system where the data dictionary is part of the DBMS) the data dictionary stores information concerning the external, conceptual, and internal levels of the database. It contains the source of each data-field value, the frequency of its use, and an audit trail concerning updates, including the who and when of each update.

Currently data dictionary systems are available as add-ons to the DBMS. Standards have yet to be evolved for integrating the data dictionary facility with the DBMS so that the two databases, one for metadata and the other for data, can be manipulated using an unified DDL/DML.

Figure 1.20 Steps in data access.



Access Aids

To improve the performance of a DBMS, a set of access aids in the form of indexes are usually provided in a database system. Commands are provided to build and destroy additional temporary indexes.

1.5.4 Database Access

Any access to the stored data is done by the data manager. The steps involved in database access can be summarized as shown in Figure 1.20.

A user's request for data is received by the data manager, which determines the physical record required. The decision as to which physical record is needed may require some preliminary consultation of the database and/or the data dictionary prior to the access of the actual data itself.

The data manager sends the request for a specific physical record to the file manager. The file manager decides which physical block of secondary storage devices contains the required record and sends the request for the appropriate block to the disk manager. A **block** is a unit of physical input/output operations between primary and secondary storage. The disk manager retrieves the block and sends it to the file manager, which sends the required record to the data manager.

1.6 Advantages and Disadvantages of a DBMS

Let us consider the pros and cons of using a DBMS.

1.6.1 Advantages of a DBMS

One of the main advantages of using a database system is that the organization can exert, via the DBA, centralized management and control over the data. The database administrator is the focus of the centralized control. Any application requiring a

Figure 1.21 Pros and cons of a DBMS.

Advantages

Centralized control
Data independence allows dynamic changes and growth potential
Data duplication eliminated with controlled redundancy
Data quality enhanced
Security enforcement possible

Disadvantages

Problems associated with centralization
Cost of software/hardware and migration
Complexity of backup and recovery

ment, and this is exacerbated in a concurrent multiuser database system. Furthermore, a database system requires a certain amount of controlled redundancies and duplication to enable access to related data items.

Centralization also means that the data is accessible from a single source, namely the database. This increases the potential severity of security breaches and disruption of the operation of the organization because of downtimes and failures. The replacement of a monolithic centralized database by a federation of independent and cooperating distributed databases resolves some of the problems resulting from failures and downtimes.

The pros and cons of a DBMS system are summarized in Figure 1.21.

1.7 Summary

Data are facts from which a conclusion can be drawn; for this reason, humans record data. Data is required in the operation of any organization, and the same or similar data may be required in various facets of its functioning.

Entity sets are the categories of objects of interest to an organization for which the organization maintains data. To store the data about an entity set, a reasonable model of the entity is made by listing the characteristics or attributes that are of relevance to the database application. In order to uniquely identify a single instance of an entity set, a primary key is devised either from the attributes that are used to model the entity set or by adding such an attribute. The values for each attribute of an instance of an entity set are grouped together and this collection is called a record type. A file is a collection of identical record type occurrences pertaining to an entity set.

A database system is an integrated collection of related files along with the details about their definition, interpretation, manipulation, and maintenance. It is an attempt to satisfy the data needs of the various applications in an organization without unnecessary duplication. The DBMS not only makes the integrated collection of reliable and accurate data available to multiple applications and users, but also exerts centralized control, prevents fraudulent or unauthorized users from accessing the data, and ensures privacy.

The DBMS provides users with a method of abstracting their data requirements and removes the drudgery of specifying the details of the storage and maintenance of data. The DBMS insulates users from changes that occur in the database. Two levels of data independence are provided by the system. Physical independence allows changes in the physical level of data storage without affecting the conceptual view. Logical independence allows the conceptual view to be changed without affecting the external view.

A DBMS is a complex software system consisting of a number of components. It provides the user with a data definition language and a data manipulation language. The user defines the external and conceptual views by using the DDL and manipulates the data contained in the database by using the DML.

The data manager is the component of the DBMS that provides an interface between the user (via the query processor or the compiled application program) and the file system. It is also responsible for controlling the simultaneous use of the database and maintaining its integrity and security. Responsibility for recovery of the database after any failure lies with the data manager.

The database administrator defines and maintains the three levels of the database as well as the mapping between levels to insulate the higher levels from changes that occur in the lower levels. The DBA is responsible for implementing measures for ensuring the security, integrity, and recovery of the database.

Key Terms

database
database system
database management system (DBMS)
entities
entity type
entity set
relationship
attributes
domain
key
primary key
superkey
candidate key
alternate key
secondary key
relationship set
record type
record occurrence
field
file
metadata
view
conceptual record
mapping
logical record
external view
user view
global view
conceptual view
internal view
external level
conceptual level
internal level
schema
external schema
conceptual schema
internal schema
conceptual database
physical database
logical data independence
physical data independence
virtual field
database administrator (DBA)
data definition language (DDL)
data dictionary
directory
system catalog
data manipulation language (DML)
query
query language
compiler
data manager
file manager
disk manager
query processor
data files
block

Exercises

1.1 Explain the differences between a file-oriented system and a database-oriented system.

1.2 Consider the application program for the support of an automatic teller machine. How does such a program communicate with the user and the database?

1.3 Define the following terms:

metadata
data independence
database administrator
query processor
data manager
external view

1.4 Give the mappings required to derive (a) the conceptual record of Figure 1.16 from the internal record of Figure 1.17, and (b) the external records of Figure 1.15 from the conceptual record of Figure 1.16.

1.5 Suppose the field EMPLOYEE.*Address* of the internal record of Figure 1.17 is replaced by the following fields:

EMPLOYEE.*Street_Number: string length* 7 *offset* 40;
EMPLOYEE.*Street: string length* 20 *offset* 47;
EMPLOYEE.*City: string length* 16 *offset* 67;
EMPLOYEE.*State: string length* 2 *offset* 83;
EMPLOYEE.*Zip: string length* 5 *offset* 85;

What changes are required in the mappings of Exercise 1.4?

1.6 Consider an airline reservation database system in which travel agents are allowed online access to make reservations on any flight. Is it possible for two travel agents located in different cities to book their respective clients the last seat on the same flight? Explain your answer.

1.7 What problems are caused by data redundancies? Can data redundancies be completely eliminated when the database approach is used? Why or why not?

1.8 Why is data important to an enterprise? How does an enterprise that has better control of its data have a competitive edge over other organizations?

1.9 Choose from the following list an enterprise you are most familiar with: college or university, public library, hospital, fast-food restaurant, department store. What are the entities of interest to this enterprise? For each such entity set, list the attributes that could be used to model each of the entities. Are there any attributes (or collections of attributes) in each entity set that would uniquely identify an instance of the entity set? What are some of the applications that may be automated using the DBMS? Design the views of these applications and the conceptual view.

1.10 Softcraft Ltd. is a corporation involved in the design, development, and marketing of software products for a family of advanced personal computers. What entities are of interest to such an enterprise? Give a list of these entities and the relationships among them.

Bibliographic Notes

Bush (Bush 45) recognized the use of the computer in the analysis of large collections of data. Fry and Sibley (Fry 76) give the historic perspectives of the evolution of DBMS systems.

The Standards Planning and Requirements Committee (SPARC) of the American National Standards Institute (ANSI) via its Committee on Computers and Information Processing (ANSI/X3) established a Study Group on Database Management Systems in 1972. Its objectives were to determine if standardization was required in database systems. An interim report (ANSI 75, ANSI 76) proposed a framework for a database management system and its interfaces. The final report (ANSI 78) gave a description in greater detail of the generalized database system architecture and identified the interfaces.

Bibliography

(ANSI 75) ANSI/X3/SPARC Study Group on DBMS, Interim Report, vol. 7, no. 2. ACM SIGMOD, 1975.

(ANSI 76) The ANSI/SPARC DBMS Model: Proc. of 2nd SHARE Working Conf. on DBMS, Montreal, 1976. D. A. Jardine (ed.). New York. North-Holland, 1977.

(ANSI 78) "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on DBMS," D. C. Tscichritzis and A. Kings (eds.). Information Systems, 1978.

(Bush 45) V. Bush, "As We May Think," *Atlantic Monthly*, July 1945, pp. 101–108.

(Cyps 78) R. J. Cypser, "Communication Architecture for Distributed Systems," Reading, MA: Addison-Wesley, 1978.

(Fry 76) J. P. Fry & E. H. Sibley, "Evolution of Data-Base Management Systems," *Computing Surveys* 8(1), March 1976, pp. 7–42.

Chapter

2

Data Models

Contents

In this chapter we look at the method of representing or modeling concrete and abstract entities. We introduce the concept of association among various attributes of an entity and the relationships among these entities. We also briefly look at the data models used in database applications. They differ in the method used to represent the relationships among entities.

2.1 Introduction

A model is an abstraction process that hides superfluous details while highlighting details pertinent to the applications at hand. A **data model** is a mechanism that provides this abstraction for database applications. Data modeling is used for representing entities of interest and their relationships in the database. It allows the conceptualization of the association between various entities and their attributes. A number of models for data representation have been developed. As with programming languages, there is no one "best" choice for all applications. Most data representation models provide mechanisms to structure data for the entities being modeled and allow a set of operations to be defined on them. The models can also enforce a set of constraints to maintain the integrity of the data. These models differ in their method of representing the associations amongst entities and attributes. The main models that we will study are the hierarchical, network, and relational models. Database management systems based on these models or variations thereof, are available from various software houses and are used to maintain corporate databases. In addition to these widely used models, others, such as the entity-relationship model, have been developed by researchers.

2.2 Data Associations

Information is obtained from raw data by using the context in which the data is obtained and made available, and the applicable conventions for its usage. For example, if we want to record the phone numbers of our friends, we usually keep a list as shown in Figure 2.1a. If we had simply written the list of the phone numbers as in Figure 2.1b, we might not be able to associate a number with a given friend. The only time we sometimes note only the phone number is when it is the only one on the list and is to be used within a very short time.

The association between Bill's name and his phone number is obtained by writing the name and number on the same line, and this mechanism, a simple data structure, is used to retrieve the corresponding information. It can also be used to modify the information if Bill changes his phone number.

When a large amount of data is stored in a database, we have to formalize the storage mechanism that will be used to obtain the correct information from the data. We have to establish a means of showing the relationship among various sets of data represented in the database. A relationship between two sets, X and Y, is a correspondence or mapping between members of the sets. A possible relationship that may exist between any two sets may be one-to-one, one-to-many, or many-to-many as shown in Figure 2.2.

there could be one or more values for the attribute on the right side. The association between these attributes is one-to-many.

Consider the entity part with the attributes *Part#* and *Color*. *Part#* is a unique part number and *Color* represents the colors in which that part is available, there being a choice of one or more. In this instance the association from the attribute *Part#* to attribute *Color* is one-to-many. There could be many parts with a given color, thereby making the association between the attributes *Part#* and *Color* many-to-many. We show these associations in Figure 2.5.

Let us return to the employee entity and its attributes: *Employee_Id, Employee_Name, Address, Phone, Skill, Dependent_Name, Kinship_to_Employee, Position_Held, Position_Start_Date, Salary, Salary_Start_Date.*

There is one value for the attribute *Employee_Id* for a given instance of the entity type EMPLOYEE. It corresponds to the property that one employee is assigned a unique identifier. Similarly, there is one value for the attribute *Employee_Name* for one instance of the entity type EMPLOYEE. The value of the attribute *Employee_Name* depends on the value of the attribute *Employee_Id*. We show this dependence by the following notation:

$$Employee\_Id \rightarrow Employee\_Name$$

to indicate that the (value of the) attribute *Employee_Name* is uniquely determined by the (value of the) attribute *Employee_Id*.

There could be many values of the attribute pair *Dependent_Name, Kinship_to_Employee* for a given instance of the entity EMPLOYEE to indicate that each employee could have many dependents. The multiple values of these attribute pairs depend on the value of the attribute *Employee_Id*. We show this dependence by the following notation:

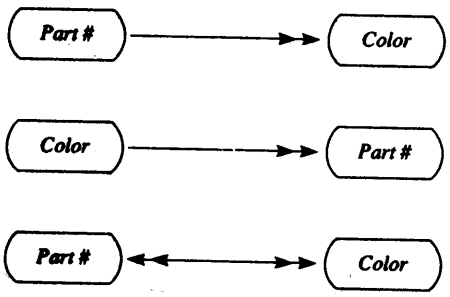
$$Employee\_Id \rightarrow\!\!\!\!\rightarrow (Dependent\_Name, Kinship\_to\_Employee)$$

Similarly, an employee could have held different positions with the organization and would have received increments in salary giving rise to the following associations from *Employee_Id:*

$$Employee\_Id \rightarrow\!\!\!\!\rightarrow (Position\_Held, Position\_Start\_Date)$$
$$Employee\_Id \rightarrow\!\!\!\!\rightarrow (Salary, Salary\_Start\_Date)$$

An employee could have had many salaries for a given position and in the event been promoted without a salary increase, could have had many positions for a given

Figure 2.5 Many-to-many association.



salary. Consequently, the association between *Position_Held* and *Salary* is many to many. We show this dependence by the following notation:

Position_Held ⟷ *Salary*

The association of these attributes is shown in Figure 2.6.

In Figure 2.7, we show the associations among the attributes of an instance of the EMPLOYEE entity. The number 12345678 identifies the employee Jill Jones, who lives at 50 Main. She has a single phone number (371-5933) and two dependents, Bill Jones, her spouse, and her son Bob Jones. She has the skills of an electrical engineer and an administrator. She was a junior engineer from December 15, 1984 and an engineer as of January 20, 1986. Her starting salary was $38,000.00 with an increment on December 15, 1985 to $39,200.00 and again on May 15, 1986 to $42,000.00.

So far, we have considered only the associations between attributes belonging to the same entity type. The definition of a given entity, however, is relative to the point of view used. [One case is illustrated with respect to the EMPLOYEE entity in Section 2.2.3 and Figure 2.16, where the attributes *(Dependent_Name, Kinship_to_Employee)* are removed from the EMPLOYEE entity and a one-to-many relationship is established.] Consequently, there could be associations between any two attributes regardless of their entities. We can approach the design of a database by considering the attributes of interest without concerning ourselves with the associated entities. We look at the associations among these attributes and design the database, grouping

Figure 2.6 Association between attributes.

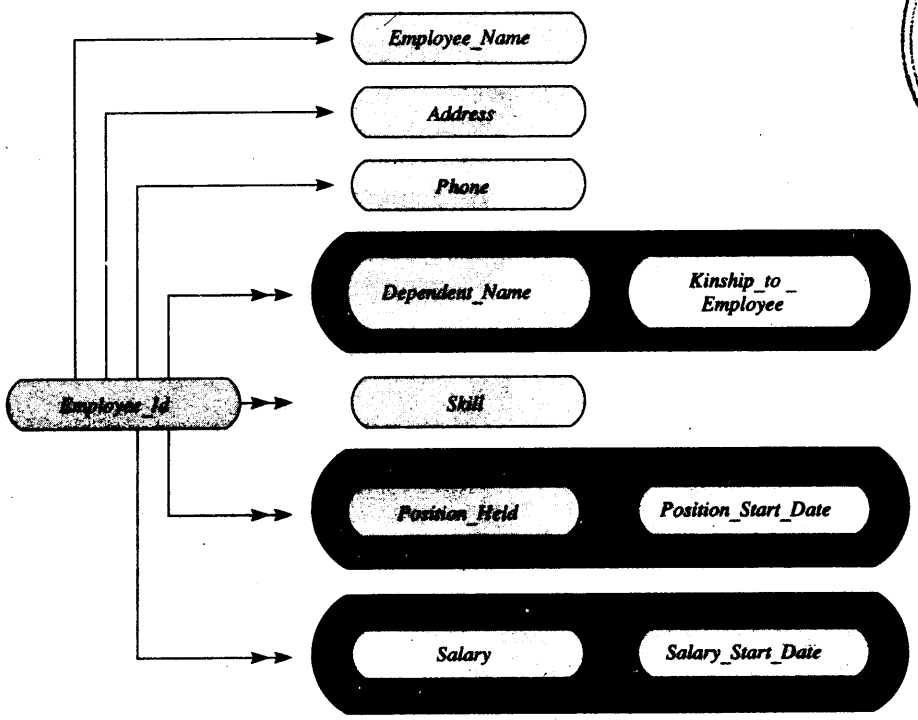
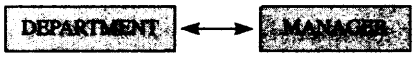


Figure 2.8 One-to-one relationship.



represented by a rectangle and the relationship between them is indicated by a direct line. The relationship from MANAGER to DEPARTMENT and from DEPARTMENT to MANAGER is both 1:1. Note that a one-to-one relationship between two entity sets does not imply that for an occurrence of an entity from one set at any time there must be an occurrence of an entity in the other set. In the case of an organization, there could be times when a department is without a manager or when an employee who is classified as a manager may be without a department to manage. Figure 2.9 shows some instances of one-to-one relationships between the entities DEPARTMENT and MANAGER. The sets of all instances of the entities are represented by the ovals.

A one-to-many relationship exists from the entity MANAGER to the entity EMPLOYEE because there are several employees reporting to the manager. As we just pointed out, there could be an occurrence of the entity type MANAGER having zero occurrences of the entity type EMPLOYEE reporting to him or her. A reverse relationship, from EMPLOYEE to MANAGER, would be many to one, since many employees may be supervised by a single manager. However, given an instance of the entity set EMPLOYEE, there could be only one instance of the entity set MANAGER to whom that employee reports (assuming that no employee reports to more than one manager). These relationships between entities are illustrated in Figure 2.10. Figure 2.11 shows some instances of these relationships.

The relationship between the entity EMPLOYEE and the entity PROJECT can be derived as follows: Each employee could be involved in a number of different projects, and a number of employees could be working on a given project. This relationship between EMPLOYEE and PROJECT is many-to-many. It is illustrated in Figure 2.12. Figure 2.13 shows some instances of such a relationship.

Figure 2.9 One-to-one relationships.

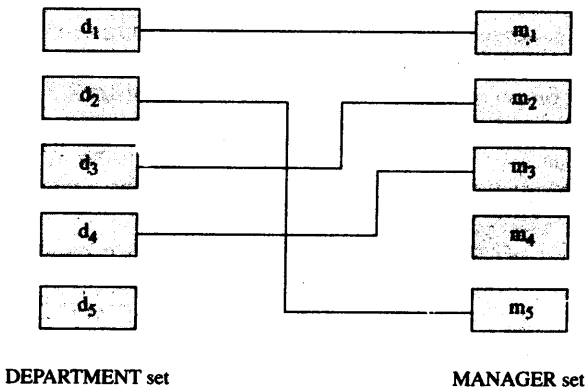


Figure 2.10 One-to-many relationship.

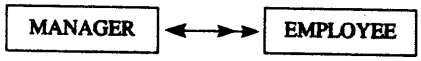


Figure 2.11 One-to-many relationships from MANAGER to EMPLOYEE and many-to-one reverse relationships.

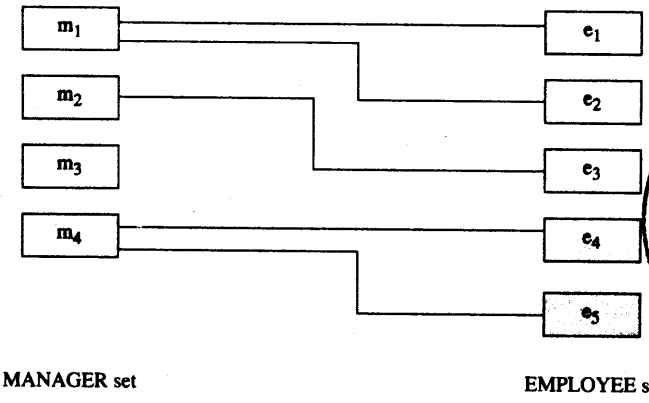


Figure 2.12 Many-to-many relationship.

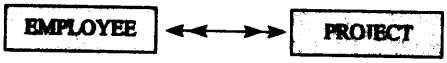
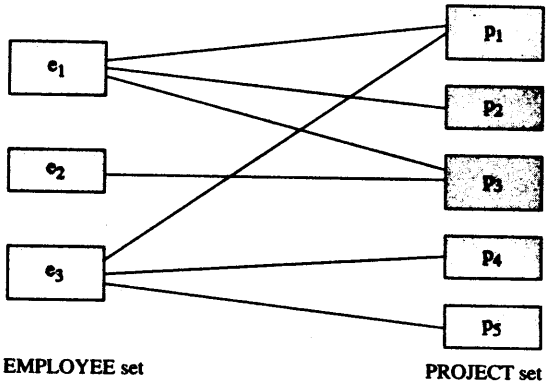


Figure 2.13 Many-to-many relationships between EMPLOYEE and PROJECT.



represent general knowledge. **Semantic data models** are able to express greater interdependencies among entities of interest. These interdependencies consist of both inclusion and exclusion, enabling the models to represent the semantics of the data in the database.

In Section 2.4 we encounter the entity-relationship data model. It provides a means for representing relationships among entities and is popular in high-level database design. Other data models in this class are beyond the scope of this text.

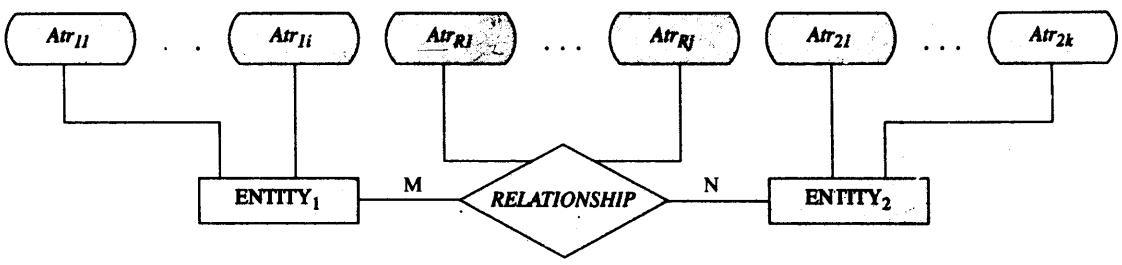
2.4 Entity-Relationship Model

The **entity-relationship (E-R) data model** grew out of the exercise of using commercially available DBMSs to model application databases. Earlier commercial systems were based on the hierarchical and network approach. The entity-relationship model is a generalization of these models. It allows the representation of explicit constraints as well as relationships. Even though the E-R model has some means of describing the physical database model, it is basically useful in the design and communication of the logical database model. In this model, objects of similar structures are collected into an entity set. The relationship between entity sets is represented by a named E-R relationship and is 1:1, 1:M, or M:N, mapping from one entity set to another. The database structure, employing the E-R model is usually shown pictorially using **entity-relationship (E-R) diagrams.** The entities and the relationships between them are shown in Figure 2.17 using the following conventions:

- An entity set is shown as a rectangle.
- A diamond represents the relationship among a number of entities, which are connected to the diamond by lines.
- The attributes, shown as ovals, are connected to the entities or relationships by lines.
- Diamonds, ovals, and rectangles are labeled. The type of relationship existing between the entities is represented by giving the cardinality of the relationship on the line joining the relationship to the entity.

Figures 2.17, 2.21, and 2.22 depict a number of entity-relationship diagrams. In Figure 2.17, the E-R diagram shows a many-to-many relationship between entities

Figure 2.17 Entity-relationship diagram.



$ENTITY_1$ and $ENTITY_2$ having the attributes (Atr_{11}, . . ., Atr_{1i}) and (Atr_{21}, . . ., Atr_{2k}), respectively. The attributes of the relationship are (Atr_{R1}, . . ., Atr_{Rj}). The relationship *ENROLLMENT* in Figure 2.21 is many to many. In Figure 2.22, the relationship *MARRIAGE* is one-to-one and *REPORTS_TO* is one-to-many.

Before discussing the E-R model in more detail, we reexamine the two components of the E-R model: entities and relationships.

2.4.1 Entities

As discussed in Chapter 1, an entity is an object that is of interest to an organization. Objects of similar types are characterized by the same set of attributes or properties. Such similar objects form an entity set or entity type. Two objects are mutually distinguishable and this fact is represented in the entity set by giving them unique identifiers.

Consider an organization such as a hotel. Some of the objects of concern to it are its employees, rooms, guests, restaurants, and menus. These collections of similar entities form the entity sets, EMPLOYEE, ROOM, GUEST_LIST, RESTAURANT, MENUS.

Given an entity set, we can determine whether or not an object belongs to it. An object may belong to more than one entity set. For example, an individual may be part of the entity set STUDENT, the entity set PART_TIME_EMPLOYEE, and the entity set PERSON. Entities interact with each other to establish relationships of various kinds.

Objects are represented by their attributes and, as objects are interdistinguishable, a subset of these attributes forms a primary key or key for uniquely identifying an instance of an entity. Entity types that have primary keys are called **strong entities.** The entity set EMPLOYEE discussed in Section 2.2 would qualify as a strong entity because it has an attribute *Employee_Id* that uniquely identifies an instance of the entity EMPLOYEE; no two instances of the entity have the same value for the attribute *Employee_Id*. Figure 2.18 shows some examples of strong entities. Only the attributes that form the primary keys are shown.

Entities may not be distinguished by their attributes but by their relationship to another entity. Recall the representation of the entity EMPLOYEE wherein the 1:M association involving the attributes *(Dependent_Name, Kinship_to_Employee)* is removed as a separate entity, DEPENDENTS. We then establish a relationship, *DEDUCTIONS*, between the modified entity EMPLOYEE* and DEPENDENTS as

Figure 2.18 Strong entities.

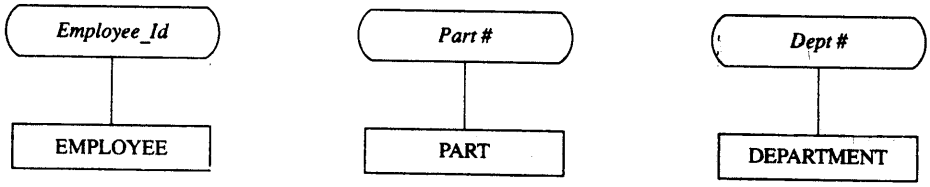


Figure 2.19 Converting an attribute association to a relationship.

shown in Figure 2.19. In this case, the instances of the entity from the set DEPENDENTS are distinguishable only by their relationship with an instance of an entity from the entity set EMPLOYEE. The relationship set *DEDUCTIONS* is an example of an **identifying relationship** and the entity set DEPENDENTS is an example of a **weak entity.**

Instances of weak entity sets associated with the same instance of the strong entity must be distinguishable from each other by a subset of the attributes of the weak entity (the subset may be the entire weak entity). This subset of attributes is called the **discriminator** of the weak entity set. For instance, the EMPLOYEE 12345678 (Jill Jones) in Figure 2.7 has two DEPENDENTS, Bill Jones, spouse and Bob Jones, son. These are distinct and can be distinguished from each other. The organization could have another Jones in its employ (with given name Jim and *Employee_Id* = 12345679), who has dependents Lydia Jones, spouse and Bob Jones, son. This is illustrated in Figure 2.20. Note also that by adding attributes such as *Social_Security_Number* of the dependent to the weak entity it can be converted into a strong entity set. However, there may be no need to do so in a given application if there is an identifying relationship.

The two instances (Bob Jones, son) of the weak entity set DEPENDENTS associated with different instances of the strong entity set EMPLOYEE are not distinguishable from each other. They are nonetheless distinct because they are associated with different instances of the strong entity set EMPLOYEE. The primary key of a weak entity set is thus formed by using the primary key of the strong entity set to which it is related, along with the discriminator of the weak entity. We rule out the case where a dependent such as Bob Jones is the son of two different employees, namely his mother and father, since only one of them will claim him as a deduction!

Figure 2.20 Instances of a 1:M converted relationship.

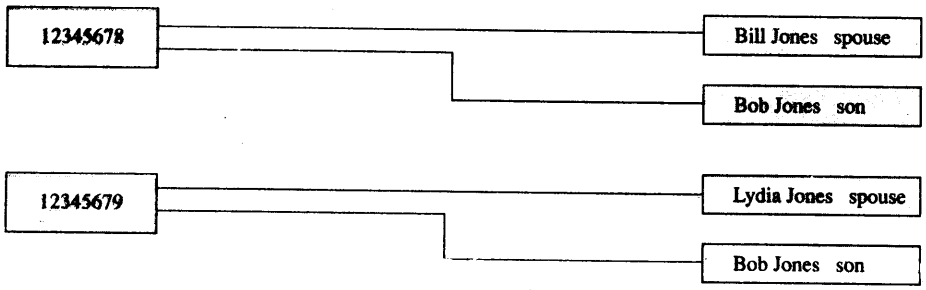
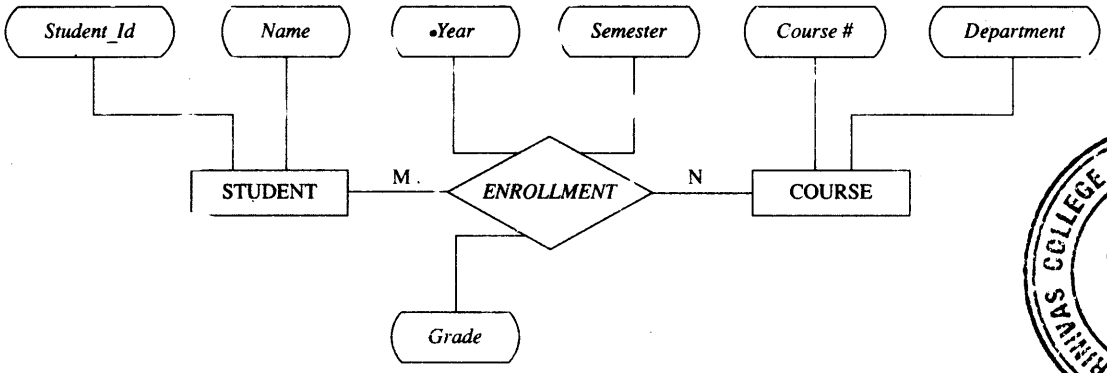


Figure 2.21 A binary relationship between different entity sets.



However, if we allow this possibility, the relationship between EMPLOYEE* and DEPENDENTS becomes many to many

2.4.2 Relationships

An association among entities is called a relationship. We looked at a relationship indirectly when we converted a 1:M association into a strong entity, a weak entity, and a relationship. A collection of relationships of the same type is called a **relationship set.** A relationship is a binary relationship if the number of entity sets involved in the relationship is two. In Figure 2.21, *ENROLLMENT* is an example of a binary relationship involving two distinct entity sets. However, the entities need not be from distinct entity sets. Figure 2.22 illustrates binary relationships that involve the same entity sets. A marriage, for example, is a relation between a man and woman that is modeled by a relationship set *MARRIAGE* between two instances of entities derived from the entity set PERSON.

A relationship that involves N entities is called an **N-ary relationship.** In Figure 2.23, *COMPUTING* is an example of a **ternary relationship** involving three entity sets. *COMPUTING* represents the relationship involving a student using a particular computing system to do the computations for a given course.

Figure 2.22 Binary relationships involving the same entity sets.

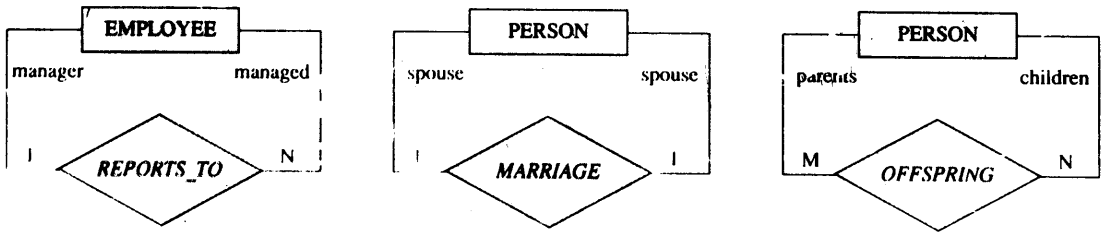
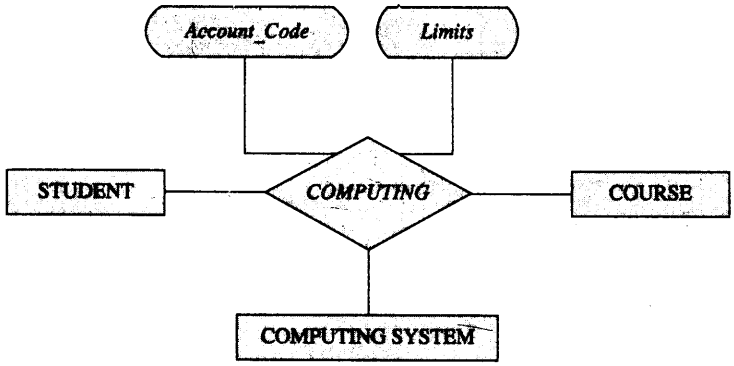


Figure 2.23 A ternary relationship.



A relationship set or simply a relationship is formally defined as follows:

Definition:

Given the entity sets $E_1, E_2, \ldots, E_k$, not necessarily distinct, then the relationship **R** is a subset of the set defined as:

$$R \subseteq \{ e_1, e_2, \ldots, e_k \text{ such that } e_i \in E_i, i = 1 \text{ to } k \}$$

A relationship can be characterized by a number of attributes. In the case of the relationship *MARRIAGE*, we can identify the attributes *Date_of_Marriage* and *Place_of_Marriage*. Similarly, in the many-to-many relationship ENROLLMENT of Figure 2.21, the attributes of the relationship are *Year*, *Semester*, and *Grade*. The attributes of the ternary relationship *COMPUTING* of Figure 2.23 are *Account_Code* and *Limits* to indicate the accounting code and the computing limits assigned to a specific student for a given course on a particular computing system.

In a relationship the roles of the entities are important. This is particularly significant when some of the entities in the relationship are not distinct. Consequently, in an occurrence of a relationship from the relationship set *MARRIAGE* involving two members from the entity set PERSON, the role of one of the entities is that of a husband and the role of the other is that of a wife. Another role that can be assigned in a more symmetrical manner in this relationship is that of spouse, as shown in Figure 2.22. In some relationships the roles are implied and need not be specified. For example, in the binary relationship *ASSIGNED_TO* between the entity sets EMPLOYEE and DEPARTMENT, the roles of the two entities are implicit.

Identification of a relationship is done by using the primary keys of the entities involved in it. Therefore, in the relationship **R** involving entity sets $E_1, E_2, \ldots, E_k$, having primary keys $p_1, p_2, \ldots, p_k$ respectively, the unique identifier of an instance of the relationship **R** is given by the composite attribute $(p_1, p_2, \ldots, p_k)$.