In Chapters 6 through 9 we discussed the relational data model, its languages, and a relational DBMS. We now discuss the network model, which, together with the hierarchical data model, was a major data model for implementing numerous commercial DBMS. We will discuss the hierarchical model in the next chapter. Historically, the network model structures and language constructs were defined by the CODASYL (Conference on Data Systems Languages) committee, so it is often referred to as the CODASYL network model. More recently, ANSI (American National Standards Institute) made a recommendation for a network definition language (NDL) standard [ANSI 1984].

The original network model and language were presented in the CODASYL Data Base Task Group’s 1971 report [DBTG 1971]; this is sometimes called the DBTG model. Revised reports in 1978 and 1981 incorporated more recent concepts. In this chapter, rather than concentrating on the details of a particular CODASYL report, we present the general concepts behind network-type databases and use the term network model rather than CODASYL model or DBTG model. We present the network model concepts independently of the entity-relationship, relational, or hierarchical data models. We show in Section 10.4 how a network model schema may be designed, starting from the ER model.

We will use PASCAL as the host language when we present the commands for a network database language, to be consistent with the rest of the book. The original CODASYL/DBTG report used COBOL as the host language. Regardless of the host programming language, the basic database manipulation commands of the network model remain the same.

In Section 10.1 we discuss record types and set types, which are the two main data-structuring constructs in the network model. Section 10.2 discusses network model constraints, and Section 10.3 presents a data definition language (DDL) for the network model. Section 10.4 shows how a network schema can be designed by mapping a con-
ceptual ER schema into the network model. In Section 10.5 we present a data manip-
ulation language for network databases which is a record-at-a-time language. Such
languages contrast with the high-level relational languages discussed in Chapters 6
through 9, which specify a set of records for retrieval in a single command. Traditionally,
the network and hierarchical models are associated with low-level record-at-a-time lan-
guages. Section 10.6 gives an overview of the IDMS commercial network DBMS.

It is possible to skip some or all of Sections 10.4 through 10.6 if a less detailed
presentation of the network model is desired.

10.1 Network Database Structures

There are two basic data structures in the network model: records and sets. We discuss
records and record types in Section 10.1.1. In Section 10.1.2 we introduce sets and their
basic properties. Section 10.1.3 presents special types of sets. We show how sets are re-
presented and implemented in Section 10.1.4. Finally, we show how 1:1 and M:N rela-
tionships are represented in the network model in Section 10.1.5.

10.1.1 Records, Record Types, and Data Items

Data is stored in records; each record consists of a group of related data values. Records
are classified into record types, where each record type describes the structure of a group
of records that store the same type of information. We give each record type a name,
and we also give a name and format (data type) for each data item (or attribute) in the
record type. Figure 10.1 shows a record type STUDENT with data items NAME, SSN,
ADDRESS, MAJORDEPT, and BIRTHDATE. The format (or data type) of each data item is
also shown in Figure 10.1.

The network model allows complex data items to be defined. A vector is a data
item that may have multiple values in a single record.* A repeating group allows inclu-
sion of a set of composite values for a data item in a single record.** For example, if we
want to include the transcript of each student within each student record, we can define
a TRANSCRIPT repeating group for the student record; TRANSCRIPT consists of the four
data items YEAR, COURSE, SEMESTER, and GRADE, as shown in Figure 10.2. The repeating
group is not essential to the modeling capability of the network model, since we can repre-
sent the same situation with two record types and a set type (see Section 10.1.2).
Repeating groups can be nested several levels deep.

All the above types of data items are called actual data items, because their values
are actually stored in the records. Virtual (or derived) data items can also be defined.
The value of a virtual data item is not actually stored in a record; instead, it is derived
from the actual data items by using some procedure that is defined specifically for this
purpose. For example, we can declare a virtual data item AGE for the record type shown

---

*This corresponds to a simple multivalued attribute in the terminology of Chapter 3.
**This corresponds to a composite multivalued attribute in the terminology of Chapter 3.
10.1 NETWORK DATABASE STRUCTURES

<table>
<thead>
<tr>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data item name</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>CHARACTER 30</td>
</tr>
<tr>
<td>SSN</td>
<td>CHARACTER 9</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>CHARACTER 40</td>
</tr>
<tr>
<td>MAJORDEPT</td>
<td>CHARACTER 10</td>
</tr>
<tr>
<td>BIRTHDATE</td>
<td>CHARACTER 9</td>
</tr>
</tbody>
</table>

Figure 10.1 A record type STUDENT.

in Figure 10.1 and write a procedure to calculate the value of AGE from the value of the actual data item BIRTHDATE in each record.

A typical database application has numerous record types—from a few to a few hundred. To represent relationships between records, the network model provides the modeling construct called set type, which we discuss next.

10.1.2 Set Types and Their Basic Properties

A set type is a description of a 1:N relationship between two record types. Figure 10.3 shows how we represent a set type diagrammatically as an arrow. This type of diagrammatic representation is called a Bachman diagram. Each set type definition consists of three basic elements:

- A name for the set type.
- An owner record type.
- A member record type.

The set type in Figure 10.3 is called MAJOR_DEPT; DEPARTMENT is the owner record type, and STUDENT is the member record type. This represents the 1:N relationship

<table>
<thead>
<tr>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

|         | 1984 | COSC3320 | Fall | A  |
|         | 1984 | COSC3340 | Fall | A  |
|         | 1984 | MATH312  | Fall | B  |
|         | 1985 | COSC4310 | Spring| C  |
|         | 1985 | COSC4330 | Spring| B  |

Figure 10.2 A repeating group TRANSCRIPT.
between academic departments and students majoring in those departments. In the database itself, there will be many set occurrences (or set instances) corresponding to a set type. Each instance relates one record from the owner record type—a department record in our example—to the set of records from the member record type related to it—the set of student records for students who major in that department. Hence, each set occurrence is composed of:

- One owner record from the owner record type.
- A number of related member records (zero or more) from the member record type.

A record from the member record type cannot exist in more than one set occurrence of a particular set type. This maintains the constraint that a set type represents a 1:N relationship. In our example, a student record can be related to at most one major department and hence is a member of at most one set occurrence of the major department set type.

A set occurrence can be identified either by the owner record or by any of the member records. Figure 10.4 shows four set occurrences (instances) of the major department set type. Notice that each set instance must have one owner record but can have any number of member records (zero or more). Hence, we usually refer to a set instance by its owner record. The four set instances in Figure 10.4 can be referred to as the ‘Computer Science’, ‘Mathematics’, ‘Physics’, and ‘Geology’ sets. It is customary to use a different representation of a set instance (Figure 10.5) where the records of the set instance are shown linked together by pointers, which corresponds to a commonly used technique for implementing sets.

In the network model, a set instance is not identical to the concept of a set in mathematics. There are two principal differences:

- The set instance has one distinguished element—the owner record—whereas in a mathematical set there is no such distinction among the elements of a set.
- In the network model, the member records of a set instance are ordered, whereas order of elements is immaterial in a mathematical set. Hence, we can refer to the
Figure 10.4 Four set instances of the set type MAJOR_DEPT.

first, second, i\textsuperscript{th}, and last member records in a set instance. Figure 10.5 shows an alternate "linked" representation of an instance of the set MAJOR_DEPT. In Figure 10.5 the record of 'Manuel Rivera' is the first STUDENT (member) record in the 'Computer Science' set, and that of 'Kareem Rashad' is the last member record. The set of the network model is sometimes referred to as an owner-coupled set or co-set, to distinguish it from a mathematical set.

10.1.3 Special Types of Sets

Two special types of sets are allowed in the CODASYL network model: SYSTEM-owned sets and multimember sets. A third type, called a recursive set, was not allowed in the original CODASYL report. We discuss these three special types of sets next.
System-owned (Singular) Sets. A system-owned set is a set with no owner record type; instead, the system* is the owner. We can think of the system as a special "virtual" owner record type with only a single record occurrence. System-owned sets serve two main purposes in the network model:

- They provide entry points into the database via the records of the specified member record type. Processing can commence by accessing members of that record type, and then retrieving related records via other sets.
- They can be used to order the records of a given record type by using the set ordering specifications. By specifying several system-owned sets on the same record type, a user can access its records in different orders.

A system-owned set allows the processing of records of a record type by using the regular set operations that we will discuss in Section 10.5.3. This type of set is called a singular set because there is only one set occurrence of it. The diagrammatic representation of the system-owned set ALL_DEPTS is shown in Figure 10.6(a), which allows DEPARTMENT records to be accessed in order of some field—say, NAME—with an appropriate set-ordering specification.

Multimember Sets. Multimember sets are used in instances where member records of a set may be of more than one record type. They are not supported in most commercial network DBMSs. The member records of a set occurrence of a multimember set type may include records from any combination of member record types. A multimember set using three member record types is shown in Figure 10.6(b). The constraint that each member record may appear in at most one set occurrence is still valid, to enforce the 1:N nature of the relationship.

*By system, we mean the DBMS software.
Recursive Sets. A set type in which the same record type plays the role of both owner and member is called a recursive set. An example of a recursive 1:N relationship that can be represented by a recursive set is the SUPERVISION relationship, which relates a supervisor employee to the list of employees directly under his or her supervision. In this relationship the EMPLOYEE record type plays both the roles: that of the owner record type (the supervisor employee), and that of the member record type (the supervisee employees).

Recursive sets were prohibited in the original CODASYL model because it is difficult to process them using the CODASYL data manipulation language (DML). The DML (see Section 10.5.2) assumes that a record belongs to a single set occurrence of each set type. With recursive sets, the same record can be an owner of one set occurrence and a member of another, both set occurrences being of the same set type. Because of this problem, it has become customary to represent a recursive set in the network model by creating an additional linking (or dummy) record type. The same technique is used to represent M:N relationships, as we shall see in Section 10.1.5. Figure 10.6(c) shows the representation of the SUPERVISION relationship, using two set types and a linking record type. In Figure 10.6(c) the SUPERVISOR set type is really a 1:1 relationship; that is, at most one SUPERVISION member record will exist in each SUPERVISOR set occurrence. We can think of each SUPERVISION linking record as representing an employee in the role of supervisor. The direct recursive set representation—usually prohibited in the network model—is shown in Figure 10.6(d). Most network DBMS implementations do not allow the same record type to participate as both owner and member in the same set type.
10.1.4 Stored Representations of Set Instances

A set instance is commonly represented as a ring (circular linked list) linking the owner record and all member records of the set, as shown in Figure 10.5. This is also sometimes called a circular chain. The ring representation is symmetric with respect to all records; hence, to distinguish between the owner record and the member records, the DBMS includes a special field, called the type field, that has a distinct value (assigned by the DBMS) for each record type. By examining the type field, the system can tell whether the record is the owner of the set instance or is one of the member records. This type field is hidden from the user and is used only by the DBMS.

In addition to the type field, a record type is automatically assigned a pointer field by the DBMS for each set type in which it participates as owner or member. This pointer can be considered to be labeled with the set type name to which it corresponds; hence, the system internally maintains the correspondence between these pointer fields and their set types. A pointer is usually called the NEXT pointer in a member record and the FIRST pointer in an owner record because these point to the next and first member records, respectively. In our example of Figure 10.5, each student record has a NEXT pointer to the next STUDENT record within the set occurrence. The NEXT pointer of the last member record in a set occurrence points back to the owner record. If a record of the member record type does not participate in any set instance, its NEXT pointer has a special nil pointer. If a set occurrence has an owner but no member records, the FIRST pointer points right back to the owner record itself or it can be nil.

The preceding representation of sets is one method for implementing set instances. In general, a DBMS can implement sets in various ways. However, the chosen representation must allow the DBMS to do all the following operations:

- Given an owner record, find all member records of the set occurrence.
- Given an owner record, find the first, i\(^{th}\), or last member record of the set occurrence. If no such record exists, give an indication of that fact.
- Given a member record, find the next (or previous) member record of the set occurrence. If no such record exists, give an indication of that fact.
- Given a member record, find the owner record of the set occurrence.

The circular linked list representation allows the system to do all of the preceding operations with varying degrees of efficiency. In general, a network database schema has many record types and set types, so a record type may participate as owner and member in numerous set types. For example, in the network schema that appears later as Figure 10.9, the EMPLOYEE record type participates as owner in four set types—MANAGES, IS_A_SUPERVISOR, E_WORKSON, and DEPENDENTS_OF—and participates as member in two set types—WORKS_FOR and SUPERVISEES. In the circular linked list representation, six additional pointer fields are added to the EMPLOYEE record type. However, no confusion arises, because each pointer is labeled by the system and plays the role of FIRST or NEXT pointer for a specific set type.
Other representations of sets allow more efficient implementation of some of the operations on sets noted previously. We briefly mention five of them here:

- **Doubly linked circular list representation:** Besides the NEXT pointer in a member record type, a PRIOR pointer points back to the prior member record of the set occurrence. The PRIOR pointer of the first member record can point back to the owner record.

- **Owner pointer representation:** This can be used in combination with either the linked list or the doubly linked list representation. For each set type an additional OWNER pointer is included in the member record type. The OWNER pointer points directly to the owner record of the set.

- **Contiguous member records:** Rather than being linked by pointers, the member records are actually placed in contiguous physical locations, typically following the owner record.

- **Pointer arrays:** An array of pointers is stored with the owner record. The $i^{th}$ element in the array points to the $i^{th}$ member record of the set instance. This is usually implemented in conjunction with the owner pointer.

- **Indexed representation:** A small index is kept with the owner record for each set occurrence. An index entry contains the value of a key indexing field and a pointer to the actual member record that has this field value. The index may be implemented as a linked list chained by next and prior pointers (the IDMS system allows this option; see Section 10.6).

These representations support the network DML operations with varying degrees of efficiency. Ideally, the programmer should not be concerned with how sets are implemented, but only with confirming that they are implemented correctly by the DBMS. However, in practice, the programmer can benefit from the particular implementation of sets, to write more efficient programs. Most systems allow the database designer to choose from among several options for implementing each set type, using a MODE statement to specify the chosen representation.

### 10.1.5 Using Sets to Represent 1:1 and M:N Relationships

A set type represents a 1:N relationship between two record types. This means that a record of the member record type can appear in only one set occurrence. This constraint is automatically enforced by the DBMS in the network model.

To represent a 1:1 relationship between two record types by using a set type, we must restrict each set occurrence to having a single member record. The network model does not provide for automatically enforcing this constraint, so the programmer must check that the constraint is not violated every time a member record is inserted into a set occurrence.

An M:N relationship between two record types cannot be represented by a single set type. For example, consider the WORKS_ON relationship between EMPLOYEES and PROJECTS. Assume that an employee can be working on several projects simultaneously and that a project typically has several employees working on it. If we try to represent
this by a set type, neither the set type in Figure 10.7(a) nor that in Figure 10.7(b) will represent the relationship correctly. Figure 10.7(a) enforces the incorrect constraint that a PROJECT record is related to only one EMPLOYEE record, whereas Figure 10.7(b) enforces the incorrect constraint that an EMPLOYEE record is related to only one PROJECT record. Using both set types E_P and P_E simultaneously, as in Figure 10.7(c), leads to the problem of enforcing the constraint that P_E and E_P are mutually consistent inverses, plus the problem of dealing with relationship attributes.

The correct method for representing an M:N relationship in the network model is to use two set types and an additional record type, as shown in Figure 10.7(d). This additional record type—WORKS_ON, in our example—is called a linking (or dummy) record type. Each record of the WORKS_ON record type must be owned by one EMPLOYEE record through the E_W set and by one PROJECT record through the P_W set and serves to relate these two owner records. This is illustrated conceptually in Figure 10.7(e).

Figure 10.7(f) shows an example of individual record and set occurrences in the linked list representation corresponding to the schema in Figure 10.7(d). Each record of the WORKS_ON record type has two NEXT pointers: the one marked NEXT(E_W) points to the next record in an instance of the E_W set, and the one marked NEXT(P_W) points

Figure 10.7  Representing M:N relationships using sets. (a)—(c) Incorrect representations of M:N relationship. (d) Correct representation of an M:N relationship using a linking (dummy) record type.
(continued on next page)
Figure 10.7 (continued) (e) Representing some occurrences of an M:N relationship with "linking occurrences." (f) Some occurrences of the set types E_W and P_W and the linking record type WORKS_ON corresponding to the M:N relationship instances shown in Figure 10.7(e).
to the next record in an instance of the P.W set. Each WORKS_ON record relates its two owner records. Each WORKS_ON record also contains the number of hours per week that an employee works on a project. The same occurrences in Figure 10.7(f) are shown in Figure 10.7(e) by displaying the W records individually, without showing the pointers.

To find all projects that a particular employee works on, we start at the EMPLOYEE record and then trace through all WORKS_ON records owned by that EMPLOYEE, using the FIRST(E.W) and NEXT(E.W) pointers. At each WORKS_ON record in the set occurrence, we find its owner PROJECT record by following the NEXT(P.W) pointers until we find a record of type PROJECT. For example, for the E2.EMPLOYEE record, we follow the FIRST(E.W) pointer in E2 leading to W1, the NEXT(E.W) pointer in W1 leading to W2, and the NEXT(E.W) pointer in W2 leading back to E2. Hence, W1 and W2 are identified as the member records in the set occurrence of E.W owned by E2. By following the NEXT(P.W) pointer in W1, we reach P1 as its owner; and by following the NEXT(P.W) pointer in W2 (and through W3 and W4), we reach P2 as its owner. Notice that the existence of direct OWNER pointers for the P.W set in the WORKS_ON records would have simplified the process of identifying the owner PROJECT record of each WORKS_ON record.

In a similar fashion, we can find all EMPLOYEE records related to a particular PROJECT. In this case the existence of owner pointers for the E.W set would simplify processing. All this pointer tracing is done automatically by the DBMS; the programmer has DML commands for directly finding the owner or the next member, as we shall discuss in Section 10.5.3.

Notice that we could represent the M:N relationship as in Figure 10.7(a) (or 10.7(b)) if we were allowed to duplicate PROJECT (or EMPLOYEE) records. In Figure 10.7(a) a PROJECT record would be duplicated as many times as there were employees working on the project. However, duplicating records creates problems in maintaining consistency among the duplicates whenever the database is updated, and it is not recommended in general.

## 10.2 Constraints in the Network Model

In explaining the network model so far, we have already discussed "structural" constraints that govern how record types and set types are structured. In the present section we discuss "behavioral" constraints that apply to (the behavior of) the members of sets when insertion, deletion, and update operations are performed on sets. Several constraints may be specified on set membership. These are usually divided into two main categories, called **insertion options** and **retention options** in CODASYL terminology. These constraints are determined during database design by knowing how a set is required to behave when member records are inserted or when owner or member records are deleted. The constraints are specified to the DBMS when we declare the database structure, using the data definition language (see Section 10.3). Not all combinations of the constraints are possible. We first discuss each type of constraint and then give the allowable combinations.
10.2.1 Insertion Options (Constraints) on Sets

The insertion constraints—or options, in CODASYL terminology—on set membership specify what is to happen when we insert a new record in the database that is of a member record type. A record is inserted by using the STORE command (see Section 10.5.4). There are two options:

- **AUTOMATIC**: The new member record is *automatically connected* to an appropriate\(^*\) set occurrence when the record is inserted.

- **MANUAL**: The new record is not connected to any set occurrence. If desired, the programmer can explicitly (*manually*) connect the record to a set occurrence subsequently, by using the CONNECT command.

For example, consider the MAJOR_DEPT set type of Figure 10.3. In this situation we can have a STUDENT record that is not related to any department through the MAJOR_DEPT set (if the corresponding student has not declared a major). We should therefore declare the MANUAL insertion option, meaning that when a member STUDENT record is inserted in the database it is not automatically related to a DEPARTMENT record through the MAJOR_DEPT set. The database user may later insert the record "manually" into a set instance when the corresponding student declares a major department. This manual insertion is accomplished by using an update operation called CONNECT, submitted to the database system, as we shall see in Section 10.5.4.

The AUTOMATIC option for set insertion is used in situations where we want to insert a member record into a set instance automatically upon storage of that record in the database. We must specify a criterion for **designating the set instance** of which each new record becomes a member. As an example, consider the set type shown in Figure 10.8(a), which relates each employee to the set of dependents of that employee. We can declare the EMP_DEPENDENTS set type to be AUTOMATIC, with the condition that a new DEPENDENT record with a particular EMPSSN value is inserted into the set instance owned by the EMPLOYEE record with the same SSN value. The DBMS locates the EMPLOYEE record such that EMPLOYEE.SSN = DEPENDENT.EMPSSN and connects the new DEPENDENT record automatically to that set instance. Notice that the SSN field should be declared so that no two EMPLOYEE records have the same SSN; otherwise, more than one set instance is identified by the above condition. In Section 10.3.2 we discuss other criteria for automatically identifying and selecting a set occurrence.

10.2.2 Retention Options (Constraints) on Sets

The retention constraints—or options, in CODASYL terminology—specify whether a record of a member record type can exist in the database on its own or whether it must always be related to an owner as a member of some set instance. There are three retention options:

- **OPTIONAL**: A member record can exist on its own *without being* a member in any occurrence of the set. It can be connected and disconnected to set occurrences at

\(^*\)The appropriate set occurrence is determined by a specification that is part of the definition of the set type, the SET OCCURRENCE SELECTION, which we discuss in Section 10.3.2 as a part of the network DDL.
Figure 10.8 Different set options. (a) An AUTOMATIC FIXED set type EMP_DEPENDENTS. (b) An AUTOMATIC MANDATORY set type EMP_DEPT.

will by means of the CONNECT and DISCONNECT commands of the network DML (see Section 10.5.4).

- MANDATORY: A member record cannot exist on its own; it must always be a member in some set occurrence of the set type. It can be reconnected in a single operation from one set occurrence to another by means of the RECONNECT command of the network DML (see Section 10.5.4).

- FIXED: As in MANDATORY, a member record cannot exist on its own. Moreover, once it is inserted in a set occurrence, it is fixed; it cannot be reconnected to another set occurrence.

We now illustrate the differences among these options by examples showing when each option should be used. First, consider the MAJOR_DEPT set type of Figure 10.3. To provide for the situation where we may have a STUDENT record that is not related to any department through the MAJOR_DEPT set, we declare the set to be OPTIONAL. In Figure 10.8(a) EMP_DEPENDENTS is an example of a FIXED set type, because we do not expect a dependent to be moved from one employee to another. In addition, every DEPENDENT record must be related to some EMPLOYEE record at all times. In Figure 10.8(b) a MANDATORY set EMP_DEPT relates an employee to the department the
Table 10.1 Set Insertion and Retention Options

<table>
<thead>
<tr>
<th>Insertion Option</th>
<th>OPTIONAL</th>
<th>Retention Option</th>
<th>MANDATORY</th>
<th>FIXED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Application program is in charge of inserting member record into set occurrence. Can CONNECT, DISCONNECT, RECONNECT</td>
<td>Not very useful.</td>
<td>Not very useful.</td>
<td></td>
</tr>
<tr>
<td>MANUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBMS inserts a new member record into a set occurrence automatically. Can CONNECT, DISCONNECT, RECONNECT.</td>
<td>DBMS inserts a new member record into a set occurrence automatically. Can RECONNECT member to a different owner.</td>
<td>DBMS inserts a new member record into a set occurrence automatically. Cannot RECONNECT member to a different owner.</td>
<td></td>
</tr>
<tr>
<td>AUTOMATIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

employee works for. Here, every employee must be assigned to exactly one department at all times; however, an employee can be reassigned from one department to another.

In general, the MANDATORY and FIXED options are used in situations where a member record should not exist in the database without being related to an owner through some set occurrence. For FIXED, the additional requirement of never moving a member record from one set instance to another is enforced. By using an appropriate insertion/retention option, the DBA is able to specify the behavior of a set type as a constraint, which is then automatically held good by the system.

10.2.3 Combinations of Insertion and Retention Options

Not all combinations of insertion and retention options are useful. For example, FIXED and MANDATORY retention options imply that a member record should always be related to an owner, so they should be used with the AUTOMATIC insertion option. While any combination of these options is technically valid, only three combinations normally make sense, and most implementations of the network model allow only these "reasonable" combinations: AUTOMATIC-FIXED, AUTOMATIC-MANDATORY, and MANUAL-OPTIONAL.* We can also think of applications where an AUTOMATIC-OPTIONAL set might be useful—namely, when the member record is automatically connected to an owner if a particular owner is specified, but otherwise the new member record is not connected to any set instance. These combinations are summarized in Table 10.1.

*The original CODASYL DBTG report did not place these restrictions on possible combinations of options.
10.2.4 Set Ordering Options

The member records in a set instance can be ordered in various ways. Order can be based on an ordering field or controlled by the time sequence of insertion of new member records. The available options for ordering can be summarized as follows:

- Sorted by an ordering field: The values of one or more fields from the member record type are used to order the member records within each set occurrence in ascending or descending order. The system maintains the order when a new member record is connected to the set instance by automatically inserting the record in its correct position in the order.

- System default: A new member record is inserted in an arbitrary position determined by the system.

- First or last: A new member record becomes the first or last record in the set occurrence at the time it is inserted. Hence, this corresponds to having the member records in a set instance stored in chronological (or reverse chronological) order.

- Next or prior: The new member record is inserted after or before the current record of the set occurrence. This will become clearer when we discuss currency indicators in Section 10.5.1.

The desired options for insertion, retention, and ordering are specified when the set type is declared in the data definition language. Details of declaring record types and set types are discussed in Section 10.3 in connection with the network model data definition language (DDL).

10.3 Data Definition in the Network Model

After designing a network database schema, we must declare all the record types, set types, data item definitions, and schema constraints to the network DBMS. The network DDL is used for this purpose. Each network DBMS has a slightly different syntax and options included in its DDL, so rather than presenting the exact syntax of the CODASYL DBTG DDL, we will concentrate on understanding the different concepts and options available in most network DBMSs.

10.3.1 Record Type and Data Item Declarations

Network DDL declarations for the record types of the COMPANY schema shown in Figure 10.9 are shown in Figure 10.10(a). Each record type is given a name by using the RECORD NAME IS clause. A format (data type) is specified for each of its data items (fields), along with any constraints on the data items. (The f.k. and * markings in Figure 10.9 are explained in Section 10.4, when we discuss ER-to-network mapping.) The usual data types available depend on the types definable in the host programming language. We
will assume that character strings, integer numbers, and formatted numbers are available.*

To specify key constraints on fields, or on combinations of fields that cannot have the same value in more than one record of a record type, the **DUPLICATES ARE NOT ALLOWED** clause is used. For example, in Figure 10.10(a), SSN is a key of the **EMPLOYEE** record type, and the combination (ESSN, PNUMBER) is a key of the **WORKS_ON** record type. Additional constraints available on fields include a constraint on the values a numeric field can take, using the **CHECK** clause. For example, we may specify that a numeric field cannot have a value greater than some number.

---

*Formatted numbers are usually specified by two numbers (i,j), where i is the total number of digits in the number and j is the number of digits after the decimal point; they are of the same size as a character string of size i+1 (one character is needed for the decimal point). A format of (7,2) permits numbers of the form ddddd.dd, where each d stands for a decimal digit.
SCHEMA NAME IS COMPANY

RECORD NAME IS EMPLOYEE
   DUPLICATES ARE NOT ALLOWED FOR SSN
   DUPLICATES ARE NOT ALLOWED FOR FNAME, MINIT, LNAME
   FNAME      TYPE IS      CHARACTER 15
   MINIT      TYPE IS      CHARACTER 1
   LNAME      TYPE IS      CHARACTER 15
   SSN        TYPE IS      CHARACTER 9
   BIRTHDATE  TYPE IS      CHARACTER 9
   ADDRESS    TYPE IS      CHARACTER 30
   SEX        TYPE IS      CHARACTER 1
   SALARY     TYPE IS      CHARACTER 10
   DEPTNAME   TYPE IS      CHARACTER 15

RECORD NAME IS DEPARTMENT
   DUPLICATES ARE NOT ALLOWED FOR NAME
   DUPLICATES ARE NOT ALLOWED FOR NUMBER
   NAME      TYPE IS      CHARACTER 15
   NUMBER    TYPE IS      NUMERIC INTEGER
   LOCATIONS TYPE IS      CHARACTER 15  VECTOR
   MGRSTART  TYPE IS      CHARACTER 9

RECORD NAME IS PROJECT
   DUPLICATES ARE NOT ALLOWED FOR NAME
   DUPLICATES ARE NOT ALLOWED FOR NUMBER
   NAME      TYPE IS      CHARACTER 15
   NUMBER    TYPE IS      NUMERIC INTEGER
   LOCATION  TYPE IS      CHARACTER 15

RECORD NAME IS WORKS_ON
   DUPLICATES ARE NOT ALLOWED FOR ESSN, PNUMBER
   ESSN      TYPE IS      CHARACTER 9
   PNUMBER   TYPE IS      NUMERIC INTEGER
   HOURS     TYPE IS      NUMERIC (4,1)

RECORD NAME IS SUPERVISOR
   DUPLICATES ARE NOT ALLOWED FOR SUPERVISOR_SSN
   SUPERVISOR_SSN  TYPE IS      CHARACTER 9

RECORD NAME IS DEPENDENT
   DUPLICATES ARE NOT ALLOWED FOR EMPSSN, NAME
   EMPSSN   TYPE IS      CHARACTER 9
   NAME     TYPE IS      CHARACTER 15
   SEX      TYPE IS      CHARACTER 1
   BIRTHDATE TYPE IS      CHARACTER 9
   RELATIONSHIP  TYPE IS      CHARACTER 10

Figure 10.10  (a) Record type declarations for the schema in Figure 10.9.
(continued on next page)
SET NAME IS ALL_DEPTS
  OWNER IS SYSTEM
  ORDER IS SORTED BY DEFINED KEYS
  MEMBER IS DEPARTMENT
  KEY IS ASCENDING NAME

SET NAME IS WORKS_FOR
  OWNER IS DEPARTMENT
  ORDER IS SORTED BY DEFINED KEYS
  MEMBER IS EMPLOYEE
    INSERTION IS MANUAL
    RETENTION IS OPTIONAL
    KEY IS ASCENDING LNAME, FNAME, MINIT
    CHECK IS DEPTNAME IN EMPLOYEE = NAME IN DEPARTMENT

SET NAME IS CONTROLS
  OWNER IS DEPARTMENT
  ORDER IS SORTED BY DEFINED KEYS
  MEMBER IS PROJECT
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY
    KEY IS ASCENDING NAME
    SET SELECTION IS BY APPLICATION

SET NAME IS MANAGES
  OWNER IS EMPLOYEE
  ORDER IS SYSTEM DEFAULT
  MEMBER IS DEPARTMENT
    INSERTION IS AUTOMATIC
    RETENTION IS MANDATORY
    SET SELECTION IS BY APPLICATION

SET NAME IS P_WORKSON
  OWNER IS PROJECT
  ORDER IS SYSTEM DEFAULT
  DUPLICATES ARE NOT ALLOWED
  MEMBER IS WORKS_ON
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED
    KEY IS ESSN
    SET SELECTION IS STRUCTURAL NUMBER IN PROJECT = PNUMBER IN WORKS_ON

SET NAME IS E_WORKSON
  OWNER IS EMPLOYEE
  ORDER IS SYSTEM DEFAULT
  DUPLICATES ARE NOT ALLOWED
  MEMBER IS WORKS_ON
    INSERTION IS AUTOMATIC
    RETENTION IS FIXED
    KEY IS PNUMBER
    SET SELECTION IS STRUCTURAL SSN IN EMPLOYEE = ESSN IN WORKS_ON

Figure 10.10  (b) Set type declarations for the schema in Figure 10.9. (continued on next page)
SET NAME IS SUPERVISEES
  OWNER IS SUPERVISOR
  ORDER IS BY DEFINED KEY
  DUPLICATES ARE NOT ALLOWED
  MEMBER IS EMPLOYEE
  INSERTION IS MANUAL
  RETENTION IS OPTIONAL
  KEY IS LNAME, MINIT, FNAME

SET NAME IS _A_SUPERVISOR
  OWNER IS EMPLOYEE
  ORDER IS SYSTEM DEFAULT
  DUPLICATES ARE NOT ALLOWED
  MEMBER IS SUPERVISOR
  INSERTION IS AUTOMATIC
  RETENTION IS MANDATORY
  KEY IS SUPERVISOR_SSN
  SET SELECTION IS BY VALUE OF SSN IN EMPLOYEE
  CHECK IS SUPERVISOR_SSN IN SUPERVISION = SSN IN EMPLOYEE

SET NAME IS DEPENDENTS_OF
  OWNER IS EMPLOYEE
  ORDER IS BY DEFINED KEY
  DUPLICATES ARE NOT ALLOWED
  MEMBER IS DEPENDENT
  INSERTION IS AUTOMATIC
  RETENTION IS FIXED
  KEY IS ASCENDING NAME
  SET SELECTION IS STRUCTURAL SSN IN EMPLOYEE = EMPSSN IN DEPENDENT

Figure 10.10 (continued) (b) Set type declarations for the schema in Figure 10.9.

10.3.2 Set Type Declarations and Set Selection Options

Figure 10.10(b) shows network DDL declarations for the set types of the COMPANY schema shown in Figure 10.9. These are more complex than record type declarations, because more options are available. Each set type is given a name by using the SET NAME IS clause. The insertion and retention options (constraints), discussed in Section 10.2, are specified for each set type by using the INSERTION IS and RETENTION IS clauses. If the insertion option is AUTOMATIC, we must also specify how the system will select a set occurrence automatically to connect a new member record to that occurrence when the record is first inserted into the database. The SET SELECTION clause is used for this purpose. Three common methods of specifying SET SELECTION are as follows:

- SET SELECTION IS STRUCTURAL: We can specify set selection by values of two fields that must match—one field from the owner record type, and one from the member record type. This is called a structural constraint in network terminology. Examples are the P_WORKSON and E_WORKSON set type declarations in Figure 10.10(b). The specified field of the owner record type must have the constraint
DUPLECTATES ARE NOT ALLOWED so that it specifies a single owner record and hence a single set occurrence.

- SET SELECTION BY APPLICATION: The set occurrence is determined via the application program, which should make the desired set occurrence the current of set (see Section 10.5.1) before the new member record is stored. The new member record is then automatically connected to the current set occurrence. An example is the MANAGES set in Figure 10.10(b); to connect an EMPLOYEE record to a DEPARTMENT as manager of that department, we must first make that EMPLOYEE record the current of set for the MANAGES set type. Storing a new DEPARTMENT record then automatically connects it to its manager EMPLOYEE record as owner.

- SET SELECTION IS BY VALUE OF <field name> IN <record type name>: A third option is to specify a field of the owner record type whose value is used to specify a set occurrence by identifying the owner record of the set. An example is the IS_A_SUPERVISOR set type declared in Figure 10.10(b), where we must set the SSN field of the UWA program variable (see Section 10.5.1) corresponding to EMPLOYEE to the value in the desired owner record before storing a new SUPERVISOR record. The field specified in the owner record type should have the constraint DUPLECTATES ARE NOT ALLOWED so that it identifies a unique owner record and hence a unique set occurrence.

Another option for sets is to specify how individual member records in a set instance will be ordered, as we discussed in Section 10.2.4. This is important because of the record-at-a-time nature of the network DML. The ORDER IS clause is used for this purpose, sometimes in conjunction with the KEY IS clause. Options for the ORDER IS clause include the following:

- ORDER IS SORTED BY DEFINED KEYS: We use the KEY IS clause to specify one or more fields from the member record type; the system uses the values of these fields to order the member records within each set instance. The KEY IS clause also specifies whether the records should be ordered in ASCENDING or DESCENDING order. An example is the WORKS_FOR set type, where EMPLOYEE records owned by a DEPARTMENT are ordered in ascending values of LNAME, FNAME, and MINIT values.

- ORDER IS FIRST (or LAST): A new member record is inserted as the first (or last) record in the set occurrence.

- ORDER IS BY SYSTEM DEFAULT: No particular order is specified on the member records in a set instance.

- ORDER IS NEXT (or PRIOR): A new member record is inserted immediately after (before) the current record of the set. The program must make the current of set (see Section 10.5.1) point to the particular record after (before) which we want the new record to be inserted in the set.

Another clause that works in conjunction with the KEY IS clause is the DUPLECTATES ARE NOT ALLOWED clause. Both clauses apply to member record types within sets. This combination specifies that no two member records within a set occurrence are allowed to have the same values for their fields declared as keys. An example is the ESSN field that is declared to be KEY for the P_WORKSON set type, meaning that no two WORKS_ON
records within the same set occurrence of P_WORKSON have the same value for ESSN. This is specified because we do not want to connect the same employee twice as a worker on the same project.

Finally, let us consider the CHECK clause, which is used to specify a structural constraint between the owner and the member records within a set occurrence. This is used with MANUAL sets to specify a condition that some fields of a member record must have the same values as some fields of the owner record. If an attempt is made to connect a member record that does not satisfy the condition of the CHECK clause, the system generates an exception condition and does not connect the member record. An example is given in the declaration of the WORKS_FOR set type in Figure 10.10(b). This constraint is similar to the SET SELECTION IS STRUCTURAL, which is used for AUTOMATIC sets.

10.4 Network Database Design Using ER-to-Network Mapping*

We now show how a conceptual database design specified as an ER schema (see Chapter 3) can be mapped to a network schema. We use the COMPANY ER schema shown in Figure 3.2 to illustrate our discussion. In a network schema we can explicitly represent a relationship type as a set type if it is 1:N; however, no explicit representation exists if it is 1:1 or M:N. One simple method of representing a 1:1 relationship type is to use a set type but to make each set instance have at most one member record. This constraint must be enforced by the programs that update the database, since the DBMS itself does not enforce it. For M:N relationship types, the standard representation is to use two set types and a linking record type. The network model allows vector fields and repeating groups, which can be used directly to represent composite and multivalued attributes or even weak entity types, as we shall see.

The COMPANY network schema shown in Figure 10.9 can be derived from the ER schema of Figure 3.2 by the following general mapping procedure. We illustrate each step by using examples from the COMPANY schema.

STEP 1: Regular Entities: For each regular entity type E in the ER schema, create a record type R in the network schema. All simple (or composite) attributes of E are included as simple (or composite) fields of R. All multivalued attributes of E are included as vector fields or repeating groups of R.

In our example we create the record types EMPLOYEE, DEPARTMENT, and PROJECT and include all their fields shown in Figure 10.9 except the fields marked by an f.k. (foreign key) or a * (relationship attribute). Notice that the LOCATIONS field of the DEPARTMENT record type is a vector field because it represents a multivalued attribute.

STEP 2: Weak Entities: For each weak entity type WE with the owner identifying entity type IE, either (a) create a record type W to represent WE, making W the member record type in a set type that relates W to the record type representing IE as owner, or (b) make a repeating group in the record type representing IE to represent the attributes of WE. If the first alternative is chosen, the key field of the record type representing IE in W can be repeated.
In Figure 10.9 we choose the first alternative; a record type DEPENDENT is created and made the member record type of the DEPENDENTS_OF set type, which is owned by EMPLOYEE. We duplicated the SSN key of EMPLOYEE in DEPENDENT and called it EMPSSN.

**STEP 3: One-to-One and One-to-Many Relationships:** For each nonrecursive binary 1:1 or 1:N relationship type R between entity types E1 and E2, create a set type relating the record types S1 and S2 that represent E1 and E2, respectively. For a 1:1 relationship type, arbitrarily choose one of S1 and S2 as owner and the other as member; however, it is preferable to choose as member a record type that represents a total participation in the relationship type. Another option for mapping a 1:1 binary relationship type R between E1 and E2 is to create a single record type S that merges E1, E2, and R and includes all their attributes; this is useful if both participations of E1 and E2 in R are total and E1 and E2 do not participate in numerous other relationship types.

For a 1:N relationship type, the record type S1 that represents the entity type E1 at the 1-side of the relationship type is chosen as owner, and the record type S2 that represents the entity type E2 at the N-side of the relationship type is chosen as member. Any attributes of the relationship type R are included as fields in the member record type S2.

In general, we can arbitrarily duplicate one (or more) attributes of an owner record type of a set type—whether it represents a 1:1 or a 1:N relationship—in the member record type. If the duplicated attribute is a unique key attribute of the owner, it can be used to declare a structural constraint on the set type or to specify automatic owner selection on set membership, as discussed in Section 10.2.1.

In our example we represent the 1:1 relationship type MANAGES from Figure 3.2 by the set type MANAGES, and we choose DEPARTMENT as member record type because of its total participation. The StartDate attribute of MANAGES becomes a field MORSTART of the member record type DEPARTMENT. The two nonrecursive 1:N relationship types from Figure 3.2, WORKS_FOR and CONTROLS, are represented by the two set types WORKS_FOR and CONTROLS in Figure 10.9. For the WORKS_ON set type, we choose to repeat a unique key field NAME of the owner record type DEPARTMENT in the member record type EMPLOYEE, and we call it DEPTNAME. We decline to repeat any key field for the CONTROLS set type. In general, a unique field from an owner record type could be repeated in the member record type.

**STEP 4: Binary Many-to-Many Relationships:** For each binary M:N relationship type R between entity types E1 and E2, create a linking record type X and make it the member record type in two set types. The set type owners are the record types S1 and S2 that represent E1 and E2. Any attributes of R are made fields of X. The designer may arbitrarily duplicate the unique (key) fields of the owner record types as fields of X.

In Figure 10.9 we create the linking record type WORKS_ON to represent the M:N relationship type WORKS_ON, and we include HOURS as its field. Two set types E_WORKSON and P_WORKSON are created with WORKS_ON as member record type. We choose to duplicate the unique key fields SSN and NUMBER of the owner record types EMPLOYEE and PROJECT in WORKS_ON, calling them ESSN and PNUMBER, respectively.

**STEP 5: Recursive Relationships:** For each recursive 1:1 or 1:N binary relationship type in which entity type E participates in both roles, create a "dummy" linking record
type D and two set types to relate D to the record type X representing E. One or both of the set types will be constrained to have set instances with a single member record—one in the case of a 1:N relationship type, and both in the case of a 1:1 relationship type.

In Figure 3.2 we have one recursive 1:N relationship type SUPERVISION. We create the dummy linking record type SUPERVISOR and the two set types IS_A_SUPERVISOR and SUPERVISEES. The IS_A_SUPERVISOR set type is constrained to have single member records by the database update programs in its set instances. We can think of each dummy SUPERVISOR member record of the IS_A_SUPERVISOR set type as representing its owner EMPLOYEE record in a supervisory role. The SUPERVISEES set type is used to relate the "dummy" SUPERVISOR record to all EMPLOYEE records that represent the employees that are his or her direct supervisees.

The preceding steps consider only binary relationship types. Step 6 shows how n-ary relationship types with \( n > 2 \) are mapped by creating a linking record type, similar to the case of an M:N relationship type.

**STEP 6: n-ary Relationships:** For each n-ary relationship type R, \( n > 2 \), create a linking record type X and make it the member record type in n set types. The owner of each set type is the record type that represents one of the entity types participating in the relationship type R. Any attributes of R are made fields of X. The designer may arbitrarily duplicate the unique (key) fields of the owner record types as fields of X.

For example, consider the relationship type SUPPLY in the ER model, as shown in Figure 10.11(a). This can be mapped to the record type SUPPLY and the three set types shown in Figure 10.11(b), where we choose not to duplicate any fields of the owners.

Notice that composite and multivalued attributes can be directly represented in the network model. In addition, we can represent weak entity types either as separate record types or as repeating groups within the owner; the latter is useful if the weak entity type does not participate in any additional relationship types. By duplicating a unique (key) field from the owner record type in the member record type, we can specify a structural constraint on set membership or automatic set selection; the DBMS will connect a member record to a set instance only if the same key field value is stored in both owner and member records. This amounts to getting the DBMS to enforce the constraint automatically. Although it is not required to duplicate a matching key field from the owner record type in the member record type, it is a recommended practice. The cost is the extra storage space required for the duplicate field in each member record. The benefits are the automatic constraint enforcement and the availability of the duplicated field in the member record without the need first to retrieve its owner.

By duplicating the key fields of owner records in member records for all set types in a network schema, we create record types that are practically identical to the relations of a relational database schema! The only differences are for recursive relationship types, multivalued attributes, and weak entity types. For recursive 1:1 or 1:N relationship types, we need not create a dummy relation in the relational schema, as we must in the network model. For weak entity types and multivalued attributes, we need not create additional record types in the network schema, as we must in the relational model. ■
10.5 Programming a Network Database*

In this section we discuss how to write programs that manipulate a network database—
including such tasks as searching for and retrieving records from the database; inserting,
deleting, and modifying records; and connecting and disconnecting records from set
occurrences. A data manipulation language is used for these purposes. The DML
associated with the network model consists of record-at-a-time commands that are embedded
in a general-purpose programming language called the host language.* In practice, the
most commonly used host languages are COBOL** and PL/I. In our examples, however,
we will write program segments in PASCAL notation augmented with network DML
commands.

*Embedded commands of the DML are also called the data sublanguage.
**The CODASYL DML in the DBTG report was originally proposed as a data sublanguage for COBOL.
10.5.1 Basic Concepts for Network Database Manipulation

To write programs for manipulating a network database, we first need to discuss some basic concepts related to how data manipulation programs are written. The database system and the host programming language are two separate software systems that are linked together by a common interface and communicate only through this interface. Because DML commands are record-at-a-time, it is necessary to identify specific records of the database as current records. The DBMS itself keeps track of a number of current records and set occurrences by means of a mechanism known as currency indicators. In addition, the host programming language needs local program variables to hold the records of different record types so that their contents can be manipulated by the host program. The set of these local variables in the program is usually referred to as the user work area (UWA). Communication between the DBMS and the host programming language is mainly accomplished through currency indicators and the user work area.

In this section we discuss these two concepts. Our examples refer to the network database schema shown in Figure 10.9, which is the network version of the COMPANY schema used in previous chapters.

The User Work Area (UWA). The UWA is a set of program variables, declared in the host program, to communicate the contents of individual records between the DBMS and the host program. For each record type in the database schema, a corresponding program variable with the same format must be declared in the program. It is customary to use the same record type names and the same field names in the UWA variables as in the database schema. In fact, the UWA variables can be automatically declared in the program by a software package that creates program variables that are equivalent to the record types declared in the DDL for a database schema.

For the COMPANY schema of Figure 10.9, if an interface between PASCAL and the network DBMS were available, it could create the PASCAL program variables shown in Figure 10.12. A single record of each record type can be copied from or written into the database by using the corresponding program variable of the UWA. The GET command (see Section 10.5.3) physically reads a record and copies it into the corresponding program variable. Then we can refer to the field values to print or to use for calculations. To write a record into the database, we first assign its field values to the fields of the program variable and then use the STORE command (see Section 10.5.3) to physically store the record in the database.

Currency Indicators. In the network DML, retrievals and updates are handled by moving or navigating through the database records; hence, keeping a trace of the search is critical. Currency indicators are a means of keeping track of the most recently accessed records and set occurrences by the DBMS. They play the role of position holders so that we may process new records starting from the ones most recently accessed until we retrieve all the records that contain the information we need. Each currency indicator can be thought of as a record pointer (or record address) that points to a single database record. In a network DBMS, several currency indicators are used:

- **Current of record type**: For each record type, the DBMS keeps track of the most recently accessed record of that record type. If no record has been accessed yet from that record type, the current record is undefined.
type LOCATIONRECORD = {'this is for the vector field LOCATIONS of DEPARTMENT'}
record
LOCATION : packed array [1..15] of char;
NEXT : ^LOCATIONRECORD
end;

var EMPLOYEE : record
FNAME : packed array [1..15] of char;
MINIT : char;
LNAME : packed array [1..15] of char;
SSN : packed array [1..9] of char;
BIRTHDATE : packed array [1..9] of char;
ADDRESS : packed array [1..30] of char;
SEX : char;
SALARY : packed array [1..10] of char;
DEPTNAME : packed array [1..15] of char
end;

DEPARTMENT : record
NAME : packed array [1..15] of char;
NUMBER : integer;
LOCATIONS : ^LOCATIONRECORD;
MGRSTART : packed array [1..9] of char
end;

PROJECT : record
NAME : packed array [1..15] of char;
NUMBER : integer;
LOCATION : packed array [1..15] of char
end;

WORKS_ON : record
ESSN : packed array [1..9] of char;
PNUMBER : integer;
HOURS : packed array [1..4] of char
end;

SUPERVISOR : record
SUPERVISOR_SSN : packed array [1..9] of char
end;

DEPENDENT : record
EMPSSN : packed array [1..9] of char;
NAME : packed array [1..15] of char;
SEX : char;
BIRTHDATE : packed array [1..9] of char;
RELATIONSHIP : packed array [1..10] of char
end;

Figure 10.12 PASCAL program variables for the UWA corresponding to the network schema in Figure 10.9.

• Current of set type: For each set type in the schema, the DBMS keeps track of the most recently accessed set occurrence from the set type. The set occurrence is specified by a single record from that set, which is either the owner or one of the member records. Hence, the current of set (or current set) points to a record, even though it is used to keep track of a set occurrence. If the program has not accessed any record from that set type, the current of set is undefined.

• Current of run unit (CRU): A run unit is a database access program that is executing (running) on the computer system. For each run unit, the CRU keeps track of the record most recently accessed by the program; this record can be from any record type in the database.
Each time a program executes a DML command, the currency indicators for the record types and set types affected by that command are updated by the DBMS. A clear understanding of how each DML command affects the currency indicators is necessary. Many DML commands both affect and depend on the currency indicators. In Section 10.5.3 we illustrate how the different DML commands affect currency indicators.

**Status Indicators.** Several status indicators return an indication of success or failure after each DML command is executed. The program can check the values of these status indicators and take appropriate action—either to continue execution or to transfer to an error-handling routine.

We call the main status variable DB_STATUS and assume that it is implicitly declared in the host program. After each DML command, the value of DB_STATUS indicates whether the command was successful or whether an error or an exception occurred. The most common exception that occurs is the END_OF_SET (EOS) exception. This is not an error; it only indicates that no more member records exist in a set occurrence. Thus it is frequently used to terminate a program loop that processes every member element of a set instance. A DML command to find the next (or prior) member of a set returns an EOS exception when no next (or prior) member record exists. The program checks for DB_STATUS = EOS to terminate the loop. We will assume that a DB_STATUS value of 0 (zero) indicates a successfully executed command with no exceptions occurring.

**Illustration of Currency Indicators and the UWA.** Suppose that a program executes database commands that result in the following events on the database instances shown in Figure 10.7(f):

- The EMPLOYEE record E3 is accessed.
- By following the FIRST(E_W) pointer in E3, the WORKS_ON record W4 is accessed; by continuing with the NEXT(E_W) pointers in WORKS_ON records, W5 and W6 are accessed.
- The record W6 is retrieved into the corresponding UWA variable.

Figure 10.13 illustrates the effects of these events on the UWA variables and the DBMS currency indicators when applied to the instances of Figure 10.7(f). Figure 10.14 shows how the currency indicators change as the events take place, with an additional event that locates the owner record P3 of W6 in the P_W set. In Figure 10.14 a pointer to a record x is shown as ^x. Notice that, after the first command, the current of set for all set types in which EMPLOYEE participates in Figure 10.9 are set to point to E3; these are E_WORKSON and DEPENDENTS_OF, which are shown in Figure 10.14, as well as SUPERVISEES, IS_A_SUPERVISOR, MANAGES, and WORKS_FOR, which are not shown. The current of EMPLOYEE record type continues to hold during the later commands, whereas that of DEPENDENT is never set (remains undefined). Notice how the currency changes for the set type E_W as we move from owner to member and for P_W as we move from member to owner.
Figure 10.13 UWA variables and currency indicators.

Figure 10.14 How currency indicators change.
10.5.2 Network Data Manipulation Language (DML)

The commands for manipulating a network database are called the network DML. These commands are typically embedded in a general-purpose programming language, called the host programming language. The DML commands can be grouped into navigation commands, retrieval commands, and update commands. Navigation commands are used to set the currency indicators to specific records and set occurrences in the database. Retrieval commands retrieve the current record of the run unit (CRU). Update commands can be divided into two subgroups—one for updating records, and the other for updating set occurrences. Record update commands are used to store new records, delete unwanted records, and modify field values, whereas set update commands are used to connect or disconnect a member record in a set occurrence or to move a member record from one set occurrence to another. The full set of commands is summarized in Table 10.2.

We now discuss each of these DML commands and illustrate our discussion with examples that use the network schema shown in Figure 10.9 and defined by the DDL statements in Figures 10.10(a) and (b). The DML commands we present are generally based on the CODASYL DBTG proposal. We use PASCAL as the host language in our examples, but students may practice writing these programs with other host languages. The examples consist of short program segments without any variable declarations. We assume that the UWA (user work area) variables shown in Figure 10.12 have been defined elsewhere in the PASCAL program. In our programs, we prefix the DML commands with a $-sign to distinguish them from the PASCAL language statements. We write PASCAL language key words—such as if, then, while, and for—in lowercase.

In our examples we often need to assign values to fields of the PASCAL UWA variables. We use the PASCAL notation for assignment. For example, to set the FNAME and LNAME fields of the EMPLOYEE UWA variable to 'John' and 'Smith', we write:

\[
\text{EMPLOYEE.FNAME} := \text{'John'}; \quad \text{EMPLOYEE.LNAME} := \text{'Smith'};
\]

Notice that, in the COBOL programming language (for which the CODASYL DML was originally designed), the same assignments are written as:

\[
\text{MOVE 'John' TO FNAME IN EMPLOYEE} \\
\text{MOVE 'Smith' TO LNAME IN EMPLOYEE}
\]

10.5.3 DML Commands for Retrieval and Navigation

The DML command for retrieving a record is the GET command. Before the GET command is issued, the program should specify the record it wants to retrieve as the CRU, by using the appropriate navigational FIND commands. There are many variations of FIND; we will first discuss the use of FIND in locating record instances of a record type and then discuss the variations for processing set occurrences.

DML Commands for Locating Records of a Record Type. There are two main variations of the FIND command for locating a record of a certain record type and making that record the CRU and current of record type. Other currency indicators may also be
Table 10.2 Summary of Network DML Commands

(RETRIEVAL)
GET
RETRIEVE THE CURRENT OF RUN UNIT (CRU) INTO THE
CORRESPONDING USER WORK AREA (UWA) VARIABLE

(NAVIGATION)
FIND
RESET THE CURRENCY INDICATORS; ALWAYS SETS
THE CRU; ALSO SETS CURRENCY INDICATORS OF
INVOLVED RECORD TYPES AND SET TYPES. THERE ARE
MANY VARIATIONS OF FIND.

(RECORD UPDATE)
STORE
STORE THE NEW RECORD IN THE DATABASE AND
MAKE IT THE CRU
ERASE
DELETE FROM THE DATABASE THE RECORD THAT IS
THE CRU
MODIFY
MODIFY SOME FIELDS OF THE RECORD THAT IS
THE CRU

(SET UPDATE)
CONNECT
CONNECT A MEMBER RECORD (THE CRU) TO A SET
INSTANCE
DISCONNECT
REMOVE A MEMBER RECORD (THE CRU) FROM A SET
INSTANCE
RECONNECT
MOVE A MEMBER RECORD (THE CRU) FROM ONE SET
INSTANCE TO ANOTHER

affected, as we shall see. The format of these two commands is as follows, where optional
parts of the command are shown in brackets, [ . . . ]:

- FIND ANY <record type name> [ USING <field list> ]
- FIND DUPLICATE <record type name> [ USING <field list> ]

We now illustrate the use of these commands with examples. To retrieve the
EMPLOYEE record for the employee whose name is John Smith and to print out his salary,
we can write EX1:

EX1: 1  \text{EMPLOYEE.FNAME} := 'John'; \text{EMPLOYEE.LNAME} := 'Smith';
2  $\text{FIND ANY EMPLOYEE USING FNAME, LNAME};$
3  \text{\textbf{if} DB\_STATUS} = 0
4        \text{then begin}
5            $\text{GET EMPLOYEE};$
6            writeln (\text{EMPLOYEE.SALARY})
7        \text{end}
8  \text{\textbf{else writeln} ('no record found');}

The FIND ANY command finds the first record in the database of the specified <record
type name> such that the field values of the record match the values initialized
earlier in the corresponding UWA fields specified in the USING clause of the command.
In EX1, lines 1 and 2 are equivalent to saying: "Search for the first employee record that satisfies the condition FNAME = 'John' and LNAME = 'Smith' and make it the current record of the run unit (CRU)." The GET statement is equivalent to saying: "Retrieve the CRU record into the corresponding UWA program variable." In general, whenever a FIND command is used, the program should check whether it successfully located a record, by testing the value of DB_STATUS. A value of 0 means that a record was successfully located, so we write the if ... then statement starting in line 3 before issuing the GET command in line 5 of EX1.

The FIND statement not only sets the CRU but also sets additional currency indicators—namely, those for the record type whose name is specified in the command and for any set types in which that record type participates as owner or member. Hence, the preceding FIND command also sets the currency indicators for the EMPLOYEE record type and for every set type in which the located record participates as owner or member of a set occurrence. However, the GET command always retrieves the CRU, which may not be the same as the current of record type.* The IDMS system combines FIND and GET into a single command, called OBTAIN.

Two variations of EX1 are worth considering. First, if we replace line 5 by just $GET, we retrieve exactly the same result as before. The difference is that including the record type name in the GET command—as in EX1—makes the system check that the CRU is of the specified record type; if not, an error is generated and the CRU is not retrieved into the UWA variable. As a second variation, if we replace line 5 by, say, $GET DEPARTMENT, an error is generated because the record type specified in the GET command, DEPARTMENT, does not match the record type of the CRU, EMPLOYEE.

If more than one record satisfies our search and we want to retrieve all of them, we must write a looping construct in the host programming language. For example, to retrieve all EMPLOYEE records for employees who work in the Research department and to print their names, we can write EX2.

EX2: EMPLOYEE.DEPTNAME := 'Research';
$FIND ANY EMPLOYEE USING DEPTNAME;
while DB_STATUS = 0 do
  begin
    $GET EMPLOYEE;
    writeln ( EMPLOYEE.FNAME, ',', EMPLOYEE.LNAME );
    $FIND DUPLICATE EMPLOYEE USING DEPTNAME
  end;

The FIND DUPLICATE command finds the next (or duplicate) record, starting from the current record, that satisfies the search. We cannot use FIND ANY, because it always locates the first record satisfying the search. Notice that "first" and "next" records have no special meaning here, because we did not specify any order on EMPLOYEE records in the DDL of Figure 10.10(b). The system searches for EMPLOYEE records physically in the order in which they are stored. However, once all EMPLOYEE records have been checked

*A variation of the network DML has been suggested which uses the GET command to retrieve the current record of the specified record type. This makes some programs easier to write. However, most network DBMSs use the GET command to retrieve the CRU, as we discuss here.
in the while-loop, the system sets DB_STATUS to a "no more records found" exception condition and the loop terminates.

**DML Commands for Set Processing.** For set processing, we have the following variations of FIND:

- FIND (FIRST | NEXT | PRIOR | LAST | ... ) <record type name>
  WITHIN <set type name> [ USING <field names> ]

- FIND OWNER WITHIN <set type name>

Once we have established a current set occurrence of a set type, we can use the FIND command to locate various records that participate in the set occurrence. We can locate either the owner record or one of the member records and make that record the CRU. We use FIND OWNER to locate the owner record and one of FIND FIRST, FIND NEXT, FIND LAST, or FIND PRIOR to locate the first, next, last, or prior member record of the set instance, respectively.

Recall that the current of set indicator may be pointing to either the owner or to any member record of a set occurrence. The FIND OWNER, FIND FIRST, and FIND LAST commands have the same effect regardless of the particular record in the set occurrence that the current of sets points to. However, FIND NEXT and FIND PRIOR do depend on the current of set. For FIND NEXT, if the current of set is the owner, it locates the first member; if the current of set is any member record except the last member, it locates the next member record; finally, if the current of the set is the last member record in the set, it sets DB_STATUS to the EOS (end-of-set) exception. For FIND PRIOR, corresponding similar actions are taken.

The next example illustrates the use of FIND FIRST and FIND NEXT. The query is to print the names of employees who work in the Research department alphabetically by last name, which is shown in EX3. This is similar to EX2 except for the alphabetic ordering requirement. EX3 retrieves first the 'Research' DEPARTMENT record and then the member EMPLOYEE records owned by that record via the WORKS_FOR set. Recall that, in the declaration of the WORKS_FOR set type in Figure 10.10(b), we specified that the member records in each set instance of WORKS_FOR are stored by ascending value of LNAME, FNAME, and MINIT. By retrieving the EMPLOYEE member records in order, we can print the employee names alphabetically in EX3. Notice how we terminate the loop by checking DB_STATUS. Once the last member record of the set occurrence is located, the subsequent FIND NEXT command sets DB_STATUS to the EOS (end-of-set) exception.

```
EX3:  DEPARTMENT.NAME := 'Research';
$FIND ANY DEPARTMENT USING NAME;
if DB_STATUS = 0 then
  begin
  $FIND FIRST EMPLOYEE WITHIN WORKS_FOR;
  while DB_STATUS = 0 do
    begin
    $GET EMPLOYEE;
    writeln ( EMPLOYEE.LNAME, ",", EMPLOYEE.FNAME );
    FIND NEXT EMPLOYEE WITHIN WORKS_FOR
    end
  end;
```
The next example illustrates the use of FIND OWNER. The query is to print the project name, project number, and hours per week for each project that employee John Smith works on (assuming there is only one such employee). This is shown in EX4. The FIND ANY command sets the CRU as well as the current record of the EMPLOYEE record type and the current of set of the E_WORKSON set type. We then loop through each WORKS_ON member record in the current E_WORKSON set, and within each loop we find the PROJECT record that owns the WORKS_ON record via the P_WORKSON set type, using the FIND OWNER command. Note that we do not have to check DB_STATUS after the FIND OWNER command, because the retention option for the set P_WORKSON is FIXED, so every WORKS_ON record must belong to a P_WORKSON set instance:

```
EX4:   EMPLOYEE.FNAME := 'John'; EMPLOYEE.LNAME := 'Smith';
      $FIND ANY EMPLOYEE USING FNAME, LNAME;
      if DB_STATUS = 0 then
         begin
            $FIND FIRST WORKS_ON WITHIN E_WORKSON;
            while DB_STATUS = 0 do
               begin
                  $GET WORKS_ON;
                  $FIND OWNER WITHIN P_WORKSON;
                  $GET PROJECT;
                  writeln ( PROJECT.NAME, PROJECT.NUMBER,
                             WORKS_ON.HOURS );
                  $FIND NEXT WORKS_ON WITHIN E_WORKSON
               end
         end;
```

In EX3 and EX4, we processed all member records of a set instance. Alternatively, we can selectively process only the member records that satisfy some condition. If the condition is an equality comparison on one (or more) fields, we can append a USING clause to the FIND command. To illustrate this, consider the request to print the names of all employees who work full-time—40 hours per week—on the ‘ProductX’ project; this example is shown as EX5:

```
EX5:   PROJECT.NAME := 'ProductX';
      $FIND ANY PROJECT USING NAME;
      if DB_STATUS = 0 then
         begin
            WORKS_ON.HOURS := '40.0';
            $FIND FIRST WORKS_ON WITHIN P_WORKSON USING HOURS;
            while DB_STATUS = 0 do
               begin
                  $GET WORKS_ON;
                  $FIND OWNER WITHIN E_WORKSON; $GET EMPLOYEE;
                  writeln (EMPLOYEE.FNAME, EMPLOYEE.LNAME);
                  $FIND NEXT WORKS_ON WITHIN P_WORKSON USING HOURS
               end
         end;
```
In EX5, the qualification USING HOURS in FIND FIRST and FIND NEXT specifies that only the WORKS_ON records in the current set instance of P_WORKSON whose HOURS field value matches the value in WORKS_ON.HOURS of the UWA, which is set to '40.0' in the program, are found. Notice that the USING clause with FIND NEXT is used to find the next member record within the same set occurrence; when we process records of a record type regardless of the sets they belong to, we use FIND DUPLICATE rather than FIND NEXT.

If the condition that selects specific member records of a set instance involves comparison operators other than equality, such as less than or greater than, we must retrieve each member record and check whether it satisfies the condition in the host program itself. The reader should attempt to modify EX4 so that only the projects for which the WORKS_ON.HOURS value exceeds 5 are retrieved. This condition must be placed immediately after the WORKS_ON record is physically retrieved.

We use numerous embedded loops in the same program segment to process several sets. For example, consider the following query: For each department, print the department's name and its manager's name; and for each employee who works in that department, print the employee's name and the list of project names that the employee works on.

This query requires us to process the system-owned set ALL_DEPTS to retrieve DEPARTMENT records. Using the WORKS_FOR set, the program retrieves the EMPLOYEE records for each DEPARTMENT. Then, for each employee found, the E_WORKSON set is accessed to locate the WORKS_ON records. For each WORKS_ON record located, a "FIND OWNER WITHIN P_WORKSON" locates the appropriate PROJECT.

Using the Host Programming Language Facilities. Because the network DML is a record-at-a-time language, we need to use the facilities of the host programming language any time a query requires a set of records. We also need to use the host programming language to calculate functions on sets of records, such as COUNTS or AVERAGES, which must be explicitly implemented by the programmer. This compares with the easy specification of such functions in high-level languages such as SQL (Chapter 7) and QUEL (Chapter 8).

A final example illustrates how we can calculate functions such as COUNT and AVERAGE. Suppose that we want to calculate the number of employees who are supervisors in each department and their average salary; this is shown in EX6. We assume that a PASCAL function convert_to_real, which converts the string value of the SALARY field into a real number, has been declared elsewhere. We must also have program variables total_sal:real and no_of_supervisors:integer declared elsewhere to accumulate the total salary and number of supervisors in each department. In EX6, notice how we test whether an employee is a supervisor by determining whether an EMPLOYEE record participates as owner in some instance of the IS_A_SUPERVISOR set:
EX6:  \$FIND FIRST DEPARTMENT \textbf{WITHIN} ALL_DEPTS;
while DB_STATUS = 0 do
  begin
  \$GET DEPARTMENT;
  write (DEPARTMENT.NAME); (* department name *)
total_sal := 0; no_of_supervisors := 0;
  \$FIND FIRST EMPLOYEE \textbf{WITHIN} WORKS_FOR;
  while DB_STATUS = 0 do
    begin
    \$GET EMPLOYEE;
    \$FIND FIRST SUPERVISOR \textbf{WITHIN} IS_A_SUPERVISOR;
    (* employee is a supervisor if it owns a SUPERVISOR record via
    IS_A_SUPERVISOR *)
    if DB_STATUS = 0 then (* test if employee is a supervisor *)
      begin
      total_sal := total_sal + convert_to_real (EMPLOYEE.SALARY);
      no_of_supervisors := no_of_supervisors + 1
      end;
    \$FIND NEXT EMPLOYEE \textbf{WITHIN} WORKS_FOR;
    end;
  writeln('number of supervisors =', no_of_supervisors);
  writeln('average salary of supervisors =', total_sal/no_of_supervisors);
  writeln();
  \$FIND NEXT DEPARTMENT \textbf{WITHIN} ALL_DEPTS
  end;

10.5.4 \textit{DML} Commands for Updating the Database

The \textit{DML} commands for updating a network database are summarized in Table 10.2. Here, we first discuss the commands for updating records—namely the \texttt{STORE}, \texttt{ERASE}, and \texttt{MODIFY} commands. These are used to insert a new record, delete a record, and modify some fields of a record, respectively. Following this, we illustrate the commands that modify set instances, which are the \texttt{CONNECT}, \texttt{DISCONNECT}, and \texttt{RECONNECT} commands.

The \texttt{STORE} Command. The \texttt{STORE} command is used to insert a new record. Before issuing a \texttt{STORE}, we must first set up the \texttt{UWA} variable of the corresponding record type so that its fields values contain the field values of the new record. For example, to insert a new \texttt{EMPLOYEE} record for John F. Smith, we can use EX7:

EX7:  EMPLOYEE.FNAME := 'John';
EMPLOYEE.LNAME := 'Smith';
EMPLOYEE.MINIT := 'F';
EMPLOYEE.SSN := '567342739';
EMPLOYEE.ADDRESS := '40 Walcott Road, Minneapolis, Minnesota 55433';
EMPLOYEE.BIRTHDATE := '10-JAN-55';
EMPLOYEE.SEX := 'M';
EMPLOYEE.SALARY := '25000.00';
EMPLOYEE.DEPTNAME := ";
$STORE EMPLOYEE;

The result of the STORE command is insertion of the current contents of the UWA record of the specified record type into the database. In addition, if the record type is an AUTOMATIC member of a set type, the record is automatically inserted into a set instance, which is determined by the SET SELECTION declaration. The newly inserted record also becomes the CRU and the current record for its record type, as well as the current of set for any set type that has the record type as owner or member.

**Effects of SET SELECTION Options on the STORE Command.** AUTOMATIC SET SELECTION options have different effects on the execution of the STORE command. Recall that, in a set type with AUTOMATIC insertion option, a new record of the member record type must be connected to a set instance at the same time it is inserted into the database by a STORE command. We briefly discuss three of the SET SELECTION options—BY STRUCTURAL, BY APPLICATION, and BY VALUE—next.

First, we illustrate the BY STRUCTURAL option. Recall from Section 10.3 (Figure 10.10) that, in the network DDL, this option has the following format:

```
SET SELECTION IS STRUCTURAL
<data item> IN <member record type> = <data item> IN <owner record type>
```

This allows the DBMS to determine by itself the set occurrence in which a newly inserted member record is to be connected; it is illustrated by the declarations for the P_WORKSON and E_WORKSON set types in Figure 10.10(b). For example, to relate the EMPLOYEE record with SSN = '567342739', just inserted in EX7, as a 40-hour-per-week worker on the project whose project number is 55, we must create and store a new linking WORKS_ON record with the appropriate ESSN and PNUMBER values, as shown in EX8. The STORE WORKS_ON command in EX8 automatically connects the newly inserted WORKS_ON record into the E_WORKSON set instance owned by the EMPLOYEE record with SSN = '567342739' and into the P_WORKSON set instance owned by the PROJECT record with NUMBER = 55 by automatically locating these owner records and their set instances. The newly inserted record also becomes the current of set for both these set types. If either of the owner records did not exist in the database, the STORE command would generate an error and the new WORKS_ON record would not be inserted into the database.

```
EX8:   WORKS_ON.ESSN := '567342739';
       WORKS_ON.PNUMBER := 55;
       WORKS_ON.HOURS := '40.0';
       $STORE WORKS_ON;
```

In the network DDL, the BY APPLICATION option has the following format:

```
SET SELECTION IS BY APPLICATION
```
The application program is responsible for selecting the proper set occurrence before storing the new member record. For example, to insert a new PROJECT record for a project that is controlled by the Research department, we must explicitly make the Research DEPARTMENT record the current of set for CONTROLS before issuing the STORE PROJECT command.

In the network DDL, the BY VALUE option has the following format:

```
SET SELECTION IS BY VALUE OF <data item> IN <owner record type>
```

It is illustrated by the declaration of the IS_A_SUPERVISOR set type in Figure 10.10(b). In this case we just set the value of the field specified in the SET SELECTION IS BY VALUE declaration—SSN of EMPLOYEE for the IS_A_SUPERVISOR set—before issuing the STORE command. This should be a key field of the owner record type, and the DBMS uses that value to find the (unique) owner of the new record. For example, to insert a new SUPERVISOR record to correspond to the employee with SSN = '567342739', we use EX9. This provides the value of EMPLOYEE.SSN in the program, which the DBMS uses to select the appropriate owner EMPLOYEE record and to automatically connect the new SUPERVISOR record to its set instance.

Notice that we could have declared SET SELECTION BY STRUCTURAL for the IS_A_SUPERVISOR set type; in fact, this would have been more appropriate in Figure 10.10(b). However, if the SUPERVISOR_SSN field was not included in the SUPERVISOR record type, the BY STRUCTURAL option cannot be used and BY VALUE would be the most appropriate choice. In general, when a UNIQUE field value from the owner record type is duplicated in the member record type, it is most appropriate to specify SET SELECTION BY STRUCTURAL for automatic sets; otherwise, SET SELECTION must be either BY VALUE or BY APPLICATION:

```
EX9: SUPERVISOR.SUPERVISOR_SSN := '567342739';
     (* create new SUPERVISION record in UWA *)
     EMPLOYEE.SSN := '567342739';
     (* set VALUE of SSN for automatic set selection *)
     $STORE SUPERVISOR;
```

The **ERASE** and **ERASE ALL** Commands. Next, we discuss deletion of records. To delete a record from the database, we first make that record the CRU and then issue the **ERASE** command. For example, to delete the EMPLOYEE record inserted in EX7, we can use EX10:

```
EX10: EMPLOYEE.SSN := '567342793';
     $FIND ANY EMPLOYEE USING SSN;
     if DB_STATUS = 0 then $ERASE EMPLOYEE;
```

The effect of an **ERASE** command on any member records that are owned by the record being deleted is determined by the set retention option. For example, the effect of the **ERASE** command in EX10 depends on the set retention for each set type that has EMPLOYEE as an owner. If retention is **OPTIONAL**, member records are kept in the database but are disconnected from the owner record before it is deleted. If retention is **FIXED**, all member records are deleted along with their owner. Finally, if retention is **MANDATORY** and some member records are owned by the record to be deleted, the **ERASE** com-
mand is rejected and an error message is generated. We cannot delete the owner, because the member records would then have no owner, which is not permitted for a MANDATORY set. These rules are recursively applied to any additional records owned by other records whose deletion is automatically triggered by an ERASE command. Deletion can thus propagate through the database and can be very damaging if it is not used carefully.

In EX10, when we ERASE the EMPLOYEE record, all WORKS_ON and DEPENDENT records owned by it are automatically deleted, because the E_WORKSON and DEPENDENTS_OF sets have a FIXED retention. However, if that EMPLOYEE record owns a SUPERVISOR record via the IS_A_SUPERVISOR set or a DEPARTMENT record via the MANAGES set, the deletion is rejected by the system because the IS_A_SUPERVISOR and MANAGES sets have MANDATORY retention. We must first explicitly remove those member records from such MANDATORY sets before issuing the ERASE command on their owner record. If the EMPLOYEE record does not own any SUPERVISOR or DEPARTMENT records via IS_A_SUPERVISOR or MANAGES, the EMPLOYEE record is deleted.

A variation of the ERASE command, ERASE ALL, allows the programmer to remove a record and all records owned by it directly or indirectly. This means that all member records owned by the record are deleted. In addition, member records owned by any of the deleted records are also deleted, down to any number of repetitions. For example, EX11 deletes the Research DEPARTMENT record, as well as all EMPLOYEE records that are owned by that DEPARTMENT via WORKS_FOR and any PROJECT records that are owned by that DEPARTMENT via CONTROLS. In addition, any DEPENDENT, SUPERVISOR, DEPARTMENT, or WORKS_ON records owned by the deleted EMPLOYEE or PROJECT records are also deleted automatically:

```
EX11:  DEPARTMENT.NAME := 'Research';
       $FIND ANY DEPARTMENT USING NAME;
       if DB_STATUS = 0 then $ERASE ALL DEPARTMENT;
```

We can also use a looping program to delete a number of records. For example, suppose that we want to delete all employees who work for the Research department, but not the DEPARTMENT record itself; to do this we can use EX12. Notice that the CRU and the current of record type for the record just deleted point to an “empty” position where the record that was deleted used to be. That is why the FIND NEXT statement in EX12 works correctly:

```
EX12:  DEPARTMENT.NAME := 'Research';
       $FIND ANY DEPARTMENT USING NAME;
       if DB_STATUS = 0 then
           begin
               $FIND FIRST EMPLOYEE WITHIN WORKS_FOR;
               while DB_STATUS = 0 do
                   begin
                       $ERASE EMPLOYEE;
                       $FIND NEXT EMPLOYEE WITHIN WORKS_FOR
                   end
           end;
```
The **MODIFY Command**. The final command for updating records is the **MODIFY command**, which changes some field values of a record. We should take the following sequence of steps to modify field values of a record:

- Make the record to be modified the **CRU**.
- Retrieve the record into the corresponding **UWA** variable.
- Modify the desired fields in the **UWA** variable.
- Issue the **MODIFY command**.

For example, to give all employees in the Research department a 10% raise, we can use **EX13**. We assume the existence of two **PASCAL** functions—`convert_to_real` and `convert_to_string`—that have been declared elsewhere; the first converts a string value of the **SALARY** field to a real number, and the second formats a real value to the **SALARY** field format.

```
EX13:  DEPARTMENT.NAME := 'Research';
$FIND ANY DEPARTMENT USING NAME;
if DB_STATUS = 0 then
  begin
    $FIND FIRST EMPLOYEE WITHIN WORKS_FOR;
    while DB_STATUS = 0 do
      begin
        $GET EMPLOYEE;
        EMPLOYEE.SALARY := convert_to_string(convert_to_real
        (EMPLOYEE.SALARY) * 1.1);
        $MODIFY EMPLOYEE;
        $FIND NEXT EMPLOYEE WITHIN WORKS_FOR
      end
  end
```

**Commands for Updating Set Instances**. We now consider the three set update operations—**CONNECT**, **DISCONNECT**, and **RECONNECT**—which are used to insert and remove member records in set instances. The **CONNECT** command inserts a member record into a set instance. The member record should be the current of run unit and is connected to the set instance that is the current of set for the set type. For example, to connect the **EMPLOYEE** record with **SSN** = '567342793' to the **WORKS_FOR** set owned by the Research **DEPARTMENT** record, we can use **EX14**:

```
EX14:  DEPARTMENT.NAME := 'Research';
$FIND ANY DEPARTMENT USING NAME;
if DB_STATUS = 0 then
  begin
    EMPLOYEE.SSN := '567342793';
    $FIND ANY EMPLOYEE USING SSN;
```
if DB_STATUS = 0 then
    $CONNECT EMPLOYEE TO WORKS_FOR;
end;

In EX14, we first locate the Research DEPARTMENT record so that the current set of
the WORKS_FOR set type becomes the set instance owned by the Research DEPARTMENT
record. Then we locate the required EMPLOYEE record so that it becomes the CRU. Finally,
we issue a CONNECT command. Notice that the EMPLOYEE record to be connected should
not be a member of any set instance of WORKS_FOR before the CONNECT command is
issued. We must use the RECONNECT command for the latter case. The CONNECT com-
mand can be used only with MANUAL sets or with AUTOMATIC OPTIONAL sets. With
other AUTOMATIC sets, the system automatically connects a member record to a set
instance, governed by the SET SELECTION option specified, as soon as the record is stored.

The DISCONNECT command is used to remove a member record from a set instance
without connecting it to another set instance. Hence, it can be used only with OPTIONAL
sets. We make the record to be disconnected the CRU before issuing the DISCONNECT
command. For example, to remove the EMPLOYEE record with SSN = '836483873' from
the SUPERVISEES set instance of which it is a member, we use EX15:

EX15:   EMPLOYEE.SSN := '836483873';
    $FIND ANY EMPLOYEE USING SSN;
    if DB_STATUS = 0
        then $DISCONNECT EMPLOYEE FROM SUPERVISEES;

Finally, the RECONNECT command can be used with both OPTIONAL and MAN-
DATORY sets, but not with FIXED sets. The RECONNECT command moves a member rec-
ord from one set instance to another set instance of the same set type. It cannot be used
with FIXED sets because a member record cannot be moved from one set instance to
another under the FIXED constraint. Before we issue the RECONNECT command, the set
instance to which the member record is to be connected should be the current of set
for the set type, and the member record to be connected should be the CRU. To do this,
we need to use an additional phrase with the FIND command—called the RETAINING
CURRENCY phrase—which we now discuss.

The RETAINING CURRENCY Phrase. The RECONNECT command and the RETAINING
CURRENCY phrase are illustrated in the context of EX16, which removes the current
manager of the Research department and assigns the employee with SSN = '836483873'
to become its new manager. Notice that the MANAGES set is declared to be AUTOMATIC
MANDATORY, so another EMPLOYEE record currently owns the Research DEPARTMENT
record in the MANAGES set type. Before issuing the RECONNECT command, we should
make the EMPLOYEE record with SSN = '836483873' the current of set for the MANAGES
set type. We should also make the Research DEPARTMENT record into the CRU, as this
is the record to be RECONNECTED to its new manager within the set MANAGES. However,
the Research DEPARTMENT record is already a member of a different MANAGES set
instance, so when it is made the CRU it will also become the current of set for the
MANAGES set type:
EX16:  

```plaintext
EMPLOYEE.SSN := '836483873';
$FIND ANY EMPLOYEE USING SSN; (* set current of set for MANAGES *)
if DB_STATUS = 0 then
    begin
        DEPARTMENT.NAME = 'Research';
        $FIND ANY DEPARTMENT USING NAME RETAINING MANAGES CURRENCY;
        (* set CRU without changing the current of set for MANAGES *)
        if DB_STATUS = 0 then $RECONNECT DEPARTMENT WITHIN MANAGES
    end;
```

To make the record become the CRU without changing the current of set, we use the RETAINING CURRENCY phrase appended to the FIND command. In EX16, we use the RETAINING MANAGES CURRENCY phrase appended to the FIND command. This changes the CRU to the Research DEPARTMENT record but leaves the current of set of MANAGES unchanged; it remains the set instance owned by the EMPLOYEE record whose SSN is 836483873.

Notice that the record to be moved should be a member of a set instance of the same set type before the RECONNECT command is issued; otherwise, we should use the CONNECT command. A RECONNECT can be replaced by a DISCONNECT and a CONNECT for OPTIONAL sets. However, for MANDATORY sets we must use the RECONNECT command if we want to move a member record from one set instance to another, because it must remain connected to an owner at all times.

### 10.6 A Network Database System—IDMS*

#### 10.6.1 Introduction

In this section we survey a popular DBMS based on the network model called the Integrated Database Management System (IDMS). It is currently marketed by Computer Associates under the name CA-IDMS. The network model has had a number of major commercial DBMS implementations: IDS II of Honeywell, DMS II of Burroughs (now UNISYS), DMS 1100 of UNIVAC (now UNISYS), VAX-DBMS of Digital Equipment, IMAGE of Hewlett-Packard, and IDMS. Most of these systems (with the exception of the original IDS) were developed following the 1971 CODASYL DBTG report (DBTG 1971) and implemented the concepts specified in that report. The DBTG report was revised in 1978 and 1981, adding some concepts and deleting others. For example, data structure descriptions or views over multiple record types (which we did not discuss) were added in the 1978 report. LOCATION MODE for record type definition and the AREA concept from the original report (which we did not discuss either) were dropped. When we analyze any particular CODASYL system (sometimes called a DBTG system), we should typically explore most of the features of the network model. Otherwise, the CODASYL DBTG classification of the system becomes questionable. Minor variations exist, some of which can be attributed to the DBTG report updates of 1978 and 1981.
DMS II of UNISYS is not a true CODASYL DBTG implementation, since it does not follow the network data model strictly. It supports embedded record types and hence can be considered as hierarchical as well as network. The TOTAL system of CINCOM also does not follow the DBTG concepts. It represents data in terms of two types of record types, called master and variable. Relationships can be defined from any master record type to any variable record type, but no relationships are possible among variable record types. TOTAL has since been enhanced into SUPRA.

The DBTG proposed three languages:

- Schema DDL—to describe a network-structured database. This is equivalent to the ANSI/SPARC conceptual schema (Section 2.2).
- Subschema DDL—to describe the part of the database relevant to one application. This corresponds to the ANSI/SPARC external schema.
- DML—data manipulation language for processing the data defined by the preceding two languages. The DML in DBTG (1971) was proposed to be used in conjunction with COBOL and hence was called COBOL DML.

All network model implementations have their own syntax for these three languages. Our DML syntax in this chapter is quite close to that in IDMS. All systems use the concept of user work areas (UWAS) and currency indicators, as shown in Figure 10.13. Some DBMSs (including IDMS) also use a device media control language (DMCL), which defines the physical characteristics of the storage media, such as buffer sizes and page sizes to which the schema definition is mapped.

IDMS is an implementation of the CODASYL DBTG concepts originally by Cullinet Software. It is designed to run on IBM mainframes under all standard operating systems. The name of the product was officially changed to IDMS/R (IDMS/Relational) in 1983, when relational facilities were added on top of the base product. Currently, it is known as CA-IDMS and has two versions: DB and DC. In this section we concentrate on the basic network-oriented facilities that correspond to the original IDMS. IDMS is closely integrated with a dictionary product called Integrated Data Dictionary (IDD).

### 10.6.2 Basic Architecture of IDMS

The IDMS family of products provides various facilities based on the central DBMS and the IDD. The IDD stores a variety of entities (an IDD term). Basic entities include users, systems, files, data items, reports, transactions, programs, and program entry points. Teleprocessing entities include messages, screens, display formats, queues, destinations, lines, terminals, and encoding tables. A number of relationships and cross-references among these entities are also stored.

Data definition facilities include three compilers that compile the schema DDL, the subschema DDL, and the DMCL. IDMS is invoked by a CALL interface for data manipulation. Users do not code the calls in their programs (unlike with IMS). Instead, they use a set of DML statements similar to those in Table 10.2. A DML preprocessor translates the DML statements into calling sequences appropriate to the host language. IDMS provides DML facilities within the following host languages: COBOL, PL/1, and IBM assembler language. The relational extensions in IDMS/R include the Automatic System Facility
(ASF)—a menu-driven front end to the system that permits a set of form-based functions to define and manipulate relational views in the form of logical records. The Logical Record Facility (LRF) creates a view of the underlying database as virtual tables for relational processing.

Computer Associates provide a set of supporting products for use with IDMS. Besides IDD, they include the following:

- **Application generator** (Application Development System/On-Line Application Generator—ADS): With this tool, an application developer defines the application functions and responses to the dictionary. It acts as a prototype tool and enables the user to have an on-line preview of the application.

- **On-line query** (OLQ): This interface allows users to ask ad hoc queries against the database or to obtain formatted reports by using predefined queries.

- **Report writer** (CULPRIT): This is a parameter-driven report writer. It actively uses the definition stored in the dictionary to generate reports. Report definitions can be stored in the dictionary and invoked by supplying the report name and parameters.

Other products are mentioned under various boxes in the CA-IDMS Environment (see Figure 10.15).

Starting from the preceding repertoire of facilities, development of an IDMS application proceeds as follows:

1. Database schema, subschema, and so on are defined by means of interactive tools called IDD utilities.
2. The schemas are compiled by using the schema compiler.
3. A DMCL description to define the physical characteristics of the database is compiled by the DMCL compiler.
4. Subschemas for various applications are compiled by the subschema compiler. The logical records or views are a part of the subschema in IDMS/R.
5. Application source programs are written in a host language with embedded DML statements and precompiled. The precompilers record in the dictionary the operations that each program performs on specific data. This information is automatically monitored by the IDD. Hence it is called an active dictionary.

### 10.6.3 Data Definition in IDMS

The schema is defined by using the schema DDL. The online schema compiler allows a free-form syntax, incremental schema modification, and schema validation. A schema has five different parts: schema description, file description, area description, record description, and set description. We will concentrate on the last two. The record and set type definitions for defining the schema of Figure 10.9 that are shown in Figure 10.10 can be used with minor syntax variations in IDMS record definitions. We will not discuss the full syntax; instead, we just highlight the differences between IDMS's DDL and the DDL in Section 10.3. The main differences are as follows:
• A record type must be assigned to an AREA when it is defined, using the phrase "WITHIN AREA."

• A record type must have a LOCATION MODE specification.

An area is a DBTG concept that refers to a group of record types. An area is typically mapped to a physically contiguous storage space. This concept has physical overtones and compromises data independence. Hence it was removed from the 1981 DBTG report. The location mode for record type is a specification of how a new occurrence of that record should be stored and how existing occurrences should be retrieved. IDMS allows the following location modes:

• CALC—The record is stored by using a CALC key from within the record; this key is used for hashing to calculate a page address, and the record is stored in or near that page. The CALC key may be declared to be unique (DUPLICATES NOT ALLOWED).

• VIA—The VIA followed by a set type name means that a member record is stored physically as close as possible to the owner record (if they belong to the same area). If they are assigned to different areas, the member record is stored at the same relative position in the area as the owner in its area. This feature was valid up to release 10.0 of IDMS.

• VIA INDEX—In this option, available after release 10.0, a record is stored via a "system-owned index" that provides a system owner record and a $B^+$-tree index. The system owner record contains the name of the index, and it points to the $B^+$-tree, which, in turn, points to the member records.

• DIRECT—A record is placed on or near a user-specified page.
SCHEMA NAME IS COMPANY

RECORD NAME IS EMPLOYEE
LOCATION MODE IS CALC USING SSN
DUPLICATES NOT ALLOWED
WITHIN EMP_AREA
  02 FNAME   PIC X(15)
  02 MINIT   PIC X
  02 LNAME   PIC X(15)
  02 SSN     PIC 9(9)
  02 BIRTHDATE PIC X(9)
  02 ADDRESS PIC X(30)
  02 SEX     PIC X
  02 SALARY  PIC 9(10)
  02 DEPTNAME PIC X(15)

RECORD NAME IS WORKS_ON
LOCATION MODE IS VIA E_WORKSON SET
WITHIN EMP_AREA
  02 ESSN    PIC 9(9)
  02 PNUMBER PIC 999 USAGE COMP-3
  02 HOURS   PIC 99 USAGE COMP-3

SET NAME IS WORKS_FOR
OWNER IS DEPARTMENT
MEMBER IS EMPLOYEE MANUAL OPTIONAL
ORDER IS SORTED
MODE IS CHAINED
ASCENDING KEY IS LNAME, FNAME
DUPLICATES ALLOWED

Figure 10.16 Sample record and set type definitions in IDMS schema DDL.

Another option, called physical sequential, is also available. In Figure 10.16 we give sample definitions of record types EMPLOYEE and WORKS_ON from the database shown in Figure 10.9. Data items are defined in the style of COBOL. See the similarity of this definition to that in Figure 10.10. EMPLOYEE is accessed by the unique CALC key of SSN, whereas WORKS_ON is accessed by using VIA E_WORKSON set type. To allow a user to start searching a database directly at a record type, that record must have been declared with location mode CALC or DIRECT. Another option is to use a system-owned set (Section 10.1.3) with that record type as a member.

IDMS also uses the concept of database keys, which we omitted in the earlier discussion. IDMS assigns a unique identifier called a database key to each record occurrence when it is entered into the database. Its value is a 4-byte identifier containing a page number and a line number.

Set Definitions. IDMS set definitions differ from the set features described earlier in the following ways:

1. The SET SELECTION clause is missing. Thus, to insert a record automatically into the set, the program must select an appropriate set occurrence, making it current.
2. The **FIXED** set retention is not provided; only **MANDATORY** and **OPTIONAL** are allowed.

3. There is no **CHECK** facility to check that a member in a set satisfies a certain constraint before it is added into the set (see the example of **CHECK** on the **WORKS_FOR** set in Figure 10.10(b)).

4. Set definition includes the choice of implementation by means of "**MODE IS CHAINED** or **INDEXED**." The former links up the members in every set occurrence by means of a circular linked list; the latter sets up an index for each set occurrence (see Section 10.1.4).

5. It is possible to designate the order in which pointers such as next or prior pointers or owner pointers (see Section 10.1.4) should be allocated within the record. This is another example of how a low-level physical specification is done as part of the **DDL** in **IDMS**.

In Figure 10.16 we show how the set type **WORKS_FOR** in Figure 10.9 would be defined in the **DDL**.

**Subschema Definition.** A subschema in **IDMS** is a subset of the original schema obtained by omitting data items, record types, and set types. Whenever a record type is omitted, all set types in which it participates as an owner or member must be eliminated. Subschema definition has two data divisions: identification division and subschema data division. The latter specifies the areas, the record types or parts thereof, and the sets to be included. An on-line subschema compiler is available. Figure 10.17 shows a hypothetical subschema definition that includes a subset of the schema in Figure 10.9 related to employees, departments, and supervisors. Subschema definition can also include logical record and path-group statements.

There are some problems with the **ERASE** command when applied to a record in the subschema. The deletion may propagate via set membership to a number of other records that may not be part of the subschema. The subschema designer must include all such record types to which a delete may propagate when defining the subschema.

**Device Media Control Language (DMCL) Description.** The **DMCL** allows a specification of the physical storage parameters that govern the mapping of the instances of data into storage for a given schema description. It specifies buffer size in terms of number of pages, and page size in bytes; it associates area names with names of buffer pools and states the names of journal files, specifying the types of device on which journal files will reside. We do not give any details of the syntax of **DMCL** here.

### 10.6.4 Data Manipulation in **IDMS**

The data manipulation language concepts introduced in Section 10.5 are applicable with minor changes to **IDMS**. All **DML** commands in Table 10.2 are available in some form. The variations are discussed next.
ADD
SUBSCHEMA NAME IS EMP_DEPT OF SCHEMA NAME COMPANY
DMCL NAME IS ED_DMCL
PUBLIC ACCESS IS ALLOWED FOR DISPLAY
ADD
AREA NAME IS EMP_AREA
DEFAULT USAGE IS SHARED UPDATE
ADD
AREA NAME IS DEP_AREA
ADD
RECORD NAME IS EMPLOYEE
   ELEMENTS ARE ALL
ADD
RECORD NAME IS SUPERVISOR
   ELEMENTS ARE ALL
ADD
RECORD NAME IS DEPARTMENT
   ELEMENTS ARE NUMBER, NAME
ADD
SET NAME IS IS_A_SUPERVISOR
ADD
SET NAME IS SUPERVISEES
ADD
SET NAME IS MANAGES
ADD
SET NAME IS WORKS_FOR

Figure 10.17  Subschema definition for the schema in Figure 10.9 (details omitted).

Retrieval Commands

- The FIND CALC <record-type-name> is applicable when a key value is already supplied in the Calc-Key field of the UWA.

- FIND <record-type-name> DBKEY is <dbkey-value> is a way of finding a record, given the database key value. This form can be used whether or not the location mode of that record is DIRECT, but it requires that the program supply the database key value (absolute location id) of the record. This is normally done if a record that has been retrieved previously is to be retrieved again in the program. The DBKEY is saved and reused.

- For retrieving a record within a set type, or within an area, the following FIND is available:

  FIND (FIRST | NEXT | PRIOR | LAST) <record-type-name> [WITHIN <set-type-name>] | WITHIN <area-name> |

  The available types of FINDs are summarized in Figure 10.18. IDMS also allows the verb OBTAIN in place of the FIND-GET combination.

- The fourth type of FIND in Figure 10.18 can be used not only within a set type but also within an area.
1. FIND ANY (or CALC) <record-type-name>
2. FIND DUPLICATE <record-type-name>
3. FIND CURRENT [ <record-type-name> | WITHIN <set-type-name> | WITHIN <area-name> ]
4. FIND [ NEXT | PRIOR | FIRST | LAST | NTH ] <record-type-name> WITHIN <set-type-name> | WITHIN <area-name>
5. FIND <record-type-name> WITHIN <set-type-name> USING <ordering-field-name>

Figure 10.18 Available types of FIND in IDMS.

Update Commands. The STORE, ERASE, and MODIFY commands of Section 10.5.4 apply to IDMS. The IDMS ERASE has four options:

- **ERASE**—Deletes the CRU (current of run unit) record if it is not the owner of any nonempty set occurrence. The system takes into account all sets for which the record type is an owner.
- **ERASE PERMANENT**—Deletes the CRU record, together with MANDATORY member occurrences of that record in any set type. OPTIONAL member occurrences are not deleted but are disconnected.
- **ERASE SELECTIVE**—Deletes the CRU record, the MANDATORY members, and the OPTIONAL members that do not participate in any other set occurrence.
- **ERASE ALL**—Deletes the CRU record and all members, whether they are MANDATORY or OPTIONAL.

In all of these options, deletion propagates recursively—that is, as if the member record that was deleted were itself an object of the ERASE command. IDMS CONNECT and DISCONNECT work as discussed in Section 10.5.4. There is no RECONNECT in IDMS.

10.6.5 Storage of Data in IDMS

In IDMS a database is logically composed of one or more areas. Areas are made up of database pages. Each page corresponds to a physical block in a file, which makes a page the basic unit of input/output. The relationship among areas and files is many-to-many; that is, an area may be mapped to a number of files and vice versa. Notice the similarity of this relationship to spaces in DB2 (and storage groups in IMS). This correspondence is stored as a part of schema description. Files are divided into fixed-length blocks called pages. Each record in a page has a prefix containing a line# (assigned from bottom of page), a record-id, and a record length. The record also includes pointers—a minimum of one per set type in which it is an owner or a member. The database key of a record on page 1051, line 3 is considered to be (1051,3).

Sets are implemented in two ways: as linked sets (MODE IS CHAIN) or as indexed sets (MODE IS INDEXED). A forward pointer is included for a set type in its owner and member record types. The designer may also request reverse pointers (LINKED TO PRIOR),
which are assigned in both the owner and the member, and pointer to owner (LINKED TO OWNER), which is assigned to the member record.

In the indexed set representation, every set occurrence is represented by the owner and a small (local) index represented by a set of index records. Figure 10.19 shows the indexed set WORKS_FOR for the schema defined in Figure 10.16. The owner and the index records are linked along a linked list using next, prior, and owner pointers. Each member record points back to its index entry. It is not necessary that an indexed set be ordered.

The system-owned sets are maintained as indexed sets. There the set-ordering specification is used to order the values of the ordering field. Each index record contains this value and the database key value. There is one occurrence of each set, and the one index so created is equivalent to a clustering index. However, many system-owned sets can be defined with different set orderings for the same record type. The CALC option provides hashed access on a CALC key to a record type (in IMS it is available only for root records). With the VIA SET option, if that set is defined by using MODE IS INDEX and ORDER IS SORTED, not only are the member records stored close to the owner (same page or a nearby page), but also the physical order of the member records closely approximates their logical order.

The product that deals with centralized database processing (without telecommunications) is called CA-IDMS/DB. IDMS is also available, providing an integrated communications facility in the form of the product CA-IDMS/DC. Details are outside our scope here, since we have not yet discussed distributed databases and "client-server" architectures.

For compatibility with the relational model, IDMS has been enhanced to support the coexistence of both navigational and relational processing—even within the same application. It naturally involves allowing records to be treated as rows of a table, where the record type is equivalent to a table or relation definition. The querying and programming facilities allow three ways of accessing data: by using the navigational DML; by using the SQL language; and by using dynamic SQL, where the SQL query is formulated within a program at run-time. These multiple-access mechanisms are supported on top of the "logical level." The "physical level" deals with stored data in the form of files.
stored in BDAM (Basic Direct Access Method) and VSAM (Virtual Sequential Access Method).

CA-IDMS/DB SQL is based on ANSI SQL standard, level 2. The language includes GRANT and REVOKE options (see Chapter 20), besides standard SQL (discussed in Chapter 7). The product called CA Extended SQL includes more advanced facilities including dynamic SQL and data/time/substring features in SQL. We will not get into the details of processing the network model's stored databases by using the relational model, but the trend is likely to be in that direction with existing network DBMS products.

10.7 Summary

In this chapter we discussed the network model, which represents data by using the building blocks of record types and set types. Each set type defines a 1:N relationship between an owner record type and a member record type. A record type can participate as owner or member in any number of set types. This is the main distinction between set types of the network model and the parent-child relationships of the hierarchical model discussed in Chapter 11. In the hierarchical model, relationships must obey a strictly hierarchical pattern. Because of the restrictions on the different types of parent-child relationships in most hierarchical DBMSs, the network model has a better modeling capability than does the hierarchical model. The modeling capability of the network model is also superior to that of the original relational model in that it explicitly models relationships, although the incorporation of foreign keys in the relational model alleviates this weakness. The JOIN operations in the relational model actually become visible and substantial as set types in the network model.

We discussed three special types of sets. SYSTEM-owned or singular sets are used to define entry points to the database. Multimember sets are used for the case when member records can be from more than one record type. Recursive sets are sets that have the same record type participating both as owner and as member. Because of implementation difficulties, recursive sets were prohibited in the original CODASYL network model, and it is now customary to represent them by using an additional linking record type and two set types.

We then discussed the types of integrity constraints on set membership that can be specified on a network schema. These are classified into insertion options (MANUAL or AUTOMATIC), retention options (OPTIONAL, MANDATORY, or FIXED), and ordering options.

The circular linked list (or ring) representation of implementing set instances was discussed; and we also discussed other implementation options for set instances that can be used to improve the performance of the circular linked list, such as double-linking and owner pointers. Other techniques that can be used to implement sets instead of linked lists, such as contiguous storage or pointer arrays, were also briefly discussed.

M:N relationships, or relationships in which more than two record types participate, can also be represented by using a linking record type. In the case of an M:N relationship with two participating record types, two set types and a linking record type are used. In the case of an n-ary relationship in which n record types participate, one linking record type and n set types are needed. One-to-one relationships are not represented explicitly;
they can be represented as a set type, but the application programs must make sure that each set instance has at most one member record at all times.

A data definition language (DDL) was presented for the network model. We saw how record types and set types are defined, and we discussed the various SET SELECTION options that are used with AUTOMATIC sets. These options specify how the DBMS identifies the appropriate set instance of an AUTOMATIC set type where a new member record is to be connected when the record is stored in the database.

We presented the commands of a record-at-a-time data manipulation language (DML) for the network model. We saw how to write programs with embedded DML commands to retrieve information from a network database and to update the database. The FIND command is used to "navigate" through the database, setting various currency indicators, whereas the GET command is used to retrieve the CRU into the corresponding UWA program variable. Commands for inserting, deleting, and modifying records and for modifying set instances were covered, as well.

Finally, we gave an overview of the IDMS commercial network DBMS.

Review Questions

10.1. Discuss the various types of fields (data items) that can be defined for record types in the network model.

10.2. Define the following terms: set type, owner record type, member record type, set instance (set occurrence), AUTOMATIC set type, MANUAL set type, MANDATORY set type, OPTIONAL set type, FIXED set type.

10.3. How are the set instances of a set type identified?

10.4. Discuss the various constraints on set membership and the cases in which each constraint should be used.

10.5. In the circular linked list (ring) representation of set instances, how does the DBMS distinguish between member records and the owner record of a set instance?

10.6. Discuss the various methods of implementing set instances. For each method, discuss which types of set-processing FIND commands can be implemented efficiently and which cannot.

10.7. What are SYSTEM-owned (singular) set types used for?

10.8. What are multimember set types used for?

10.9. What are recursive set types? Why are they not allowed in the original CODASYL network model? How can they be implemented via a linking record type?

10.10. Show how each of the following types of relationships is represented in the network model: (a) M:N relationships; (b) n-ary relationships with n > 2; (c) 1:1 relationships. Discuss how an ER schema can be mapped into a network schema.

10.11. Discuss the following concepts, and explain what each is used for when writing a network DML database program: (a) the user work area (UWA); (b) currency indicators; (c) the database status indicator.
10.12. Discuss the different types of currency indicators, and tell how each type of navigational FIND command affects each currency indicator.

10.13. Describe the various SET SELECTION options for AUTOMATIC set types, and specify the circumstances under which each SET SELECTION option should be chosen.

10.14. State what each of the following clauses in the network DDL specifies: (a) DUPLICATES ARE NOT ALLOWED; (b) ORDER IS; (c) KEY IS; (d) CHECK.

10.15. For what purpose is the RETAINING CURRENCY clause used with the FIND command in the network DDL?

10.16. How does the ERASE ALL command differ from the ERASE command?

10.17. Discuss the CONNECT, DISCONNECT, and RECONNECT commands, and specify the types of set constraints under which each can be used.

Exercises

10.18. Specify the queries of Exercise 6.19, using network DML commands embedded in PASCAL on the network database schema of Figure 10.9. Use the PASCAL program variables declared in Figure 10.12, and declare any additional variables you may need.

10.19. How would you modify the program segment in EX3 so that it retrieves for each department the department name and the names of all employees who work in that department, ordered alphabetically?

10.20. Consider the following query: "For each department, print the department name and its manager's name; for each employee who works in that department, print the employee's name and the list of project names the employee works on." First write a program segment for this query. Then modify it so that the following conditions are met, one at a time:
   a. Only departments with greater than 10 employees are listed.
   b. Only employees who work more than 20 total hours are listed.
   c. Only employees with dependents are listed.

10.21. Consider the network database schema shown in Figure 10.20, which corresponds to the relational schema of Figure 2.1. Write appropriate network DDL statements to define the record types and set types of the schema. Choose appropriate set constraints for each set type, and justify your choices.

10.22. Write PASCAL program segments with embedded network DML commands to specify the queries of Exercise 7.16 on the schema of Figure 10.20.

10.23. Write PASCAL program segments with embedded network DML commands to do the updates and tasks of Exercises 7.17 and 7.18 on the network database schema of Figure 10.20. Specify any program variables you need.

10.24. During processing of the query "Find all courses offered by the COSC department; and for each course, list its name, its sections, and its prerequisites" on the schema of Figure 10.20, the following steps are performed by the DBMS: (a) find
the DEPARTMENT record for COSC; (b) find a COURSE record for COSC541 as a member of OFFERS; (c) find a SECTION record for section S32491 as a member of HAS_SECTIONS; (d) find a PREREQUISITE record occurrence—say, P1—as a member of IS_WITH_PREREQ; (e) find a member course record of HAS_PREREQ—say, the course MATH143. Show the currency indicators of set types, record types, and the CRU after each of these events, using the notation in Figure 10.14. Assume that all currency indicators were nil originally. What happens to the currency of the COURSE record type after step (e)? Suppose that we did FIND NEXT COURSE WITHIN OFFERS after step (e); what problem would arise, and how could we solve it? (Hint: Consider using the RETAINING CURRENCY phrase in some of the commands.)

10.25. Write procedures in pseudocode (PASCAL style) that may be part of the DBMS software, and outline the action taken by the following DML commands:

a. Process the STORE <record type> command. This should check for set type definitions in the DBMS catalog to determine which sets the record type participates in as owner or member, and take appropriate action.

b. Process the ERASE <record type> and ERASE ALL <record type> commands.

10.26. Choose some database application that you are familiar with or interested in.

a. Design a network database schema for your database application.

b. Declare your record types and set types using the network DDL.

c. Specify a number of queries and updates that are needed by your database
application, and write a PASCAL program segment with embedded network DML commands for each of your queries.

d. Implement your database if you have a network DBMS system available.

10.27. Map the following ER schemas into network schemas. Specify for each set type the insertion and retention options and any ordering options, and justify your choices.

a. The AIRLINES ER schema of Figure 3.19.

b. The BANK ER schema of Figure 3.20.

c. The SHIP_TRACKING ER schema of Figure 6.21.

10.28. Consider the network schema diagram for a LIBRARY database shown in Figure 10.21, which corresponds to the relational schema of Figure 6.22.

a. Specify appropriate PASCAL UWA variables for the LIBRARY schema.

b. Write PASCAL program segments with embedded network DML commands for each of the queries in Exercise 6.26.

c. Compare the relational (Figure 6.22) and network (Figure 10.21) schemas for the LIBRARY database. Identify their similarities and differences. How can you make the network schema more similar to the relational schema?

10.29. Try to map the network schema of Figure 10.21 into an ER schema. This is part of a process known as reverse engineering, where a conceptual schema is created for an existing implemented database. State any assumptions you make.

![Network schema diagram for a LIBRARY database.](image)

Figure 10.21 Network schema diagram for a LIBRARY database.
Selected Bibliography

Early work on the network data model was done by Charles Bachman during the development of the first commercial DBMS, IDS (Bachman and Williams 1964) at General Electric and later at Honeywell. Bachman also introduced the earliest diagrammatic technique for representing relationships in database schemas, called data structure diagrams (Bachman 1969) or Bachman diagrams. Bachman won the 1973 Turing Award, ACM's highest honor, for his work, and his Turing Award lecture (Bachman 1973) presents the view of the database as a primary resource and the programmer as a "navigator" through the database. In a 1974 debate between proponents and opponents of the relational approach, he represented the latter (Bachman 1974). Other work on the network model was performed by George Dodd (Dodd 1966) at General Motors Research. Dodd (1969) gives an early survey of database management techniques.

The DBTG (Data Base Task Group) of CODASYL (Conference on Data Systems Languages) was set up to propose DBMS standards. The DBTG 1971 report (DBTG 1971) contains schema and subschema DDLs and a DML for use with COBOL. A revised report (CODASYL 1978) was made in 1978, and another draft revision was made in 1981. The X3H2 committee of ANSI (the American National Standards Institute) proposed a standard network language called NDL.


Network database management is surveyed by Taylor and Frank (1976). Extensive treatments of the network model are offered in the books by Cardenas (1985), Kroenke and Dolan (1988), and Olle (1978). The IDMS system is described in several CA-IDMS/DB manuals published by Computer Associates.