APPENDIX E An Overview of the Network Data Model

This appendix provides an overview of the network data model.[1] The original network model and language were presented in the CODASYL Data Base Task Group's 1971 report; hence it is sometimes called the DBTG model. Revised reports in 1978 and 1981 incorporated more recent concepts. In this appendix, 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.

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. Although the network model and the object-oriented data model are both navigational in nature, the data structuring capability of the network model is much more elaborate and allows for explicit insertion/deletion/modification semantic specification. However, it lacks some of the desirable features of the object models that we discussed in Chapter 11, such as inheritance and encapsulation of structure and behavior.

1. The complete chapter on the network data model and about the IDMS system from the second edition of this book is available at http://www.awl.com/cseng/titles/0-8053-1755-4. This appendix is an edited excerpt of that chapter.

E.1 Network Data Modeling Concepts

There are two basic data structures in the network model: records and sets.

E.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 E.1 shows a record type STUDENT with data items NAME, SSN, ADDRESS, MAJORDEPT, and BIRTHDATE.

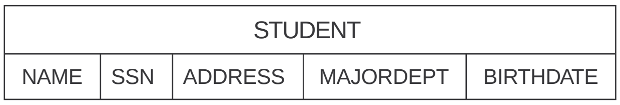
We can declare a virtual data item (or derived attribute) AGE for the record type shown in Figure E.1 and write a procedure to calculate the value of AGE from the value of the actual data item BIRTHDATE in each record.

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.

E.1.2 Set Types and Their Basic Properties

A **set type** is a description of a 1:N relationship between two record types. Figure E.2 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.



data item name	format
NAME	CHARACTER 30
SSN	CHARACTER 9
ADDRESS	CHARACTER 40
MAJORDEPT	CHARACTER 10
BIRTHDATE	CHARACTER 9

Figure E.1 A record type STUDENT.

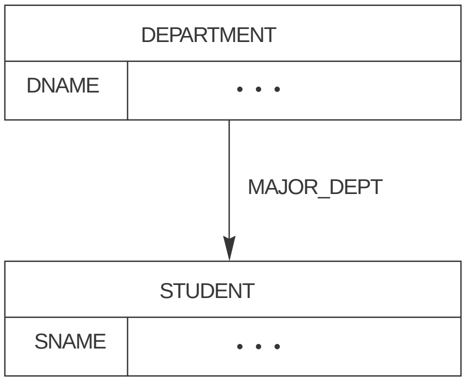


Figure E.2 The set type MAJOR_DEPT.

The set type in Figure E.2 is called MAJOR_DEPT; DEPARTMENT is the **owner** record type, and STUDENT is the **member** record type. This represents the 1:N relationship 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_DEPT set type.

A set occurrence can be identified either by the *owner record* or by *any of the member records*. Figure E.3 shows four set occurrences (instances) of the MAJOR_DEPT 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 E.3 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 E.4) where the records of the set instance are shown linked together by pointers, which corresponds to a commonly used technique for implementing sets.

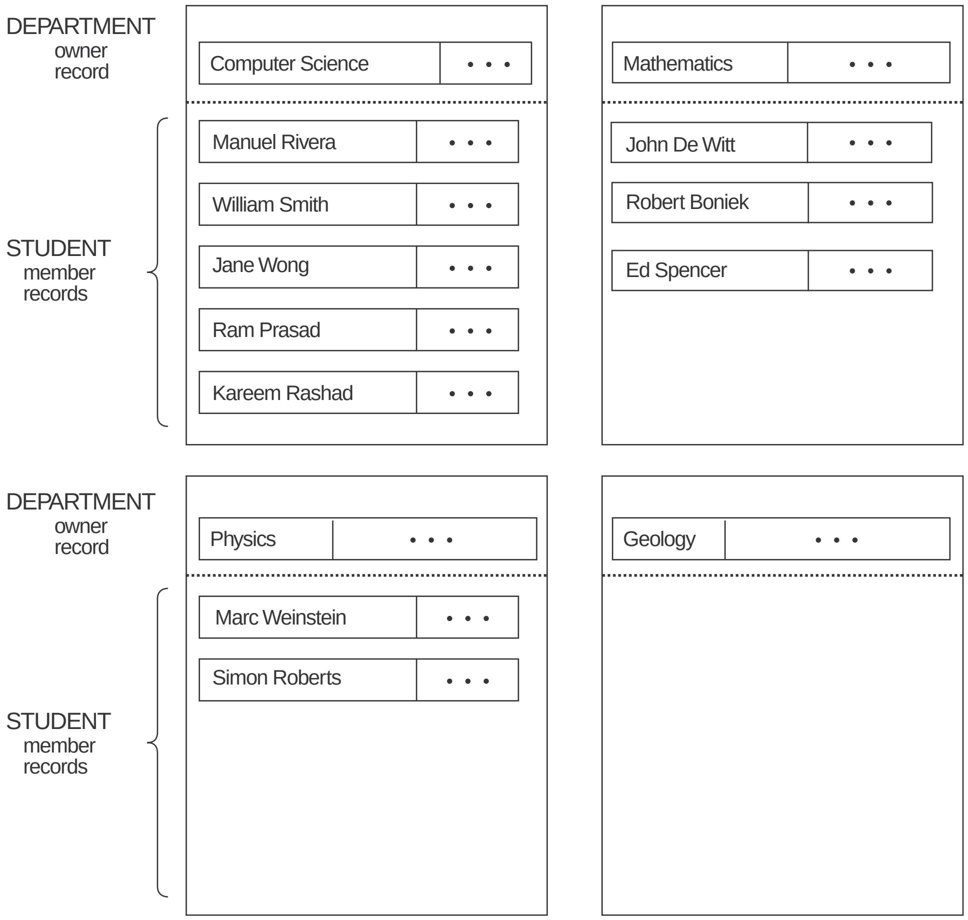


Figure E.3 Four set instances of the set type MAJOR_DEPT.

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 first, second, i^{th}, and last member records in a set instance. Figure E.4 shows an alternate "linked" representation of an instance of the set MAJOR_DEPT. In Figure E.4 the

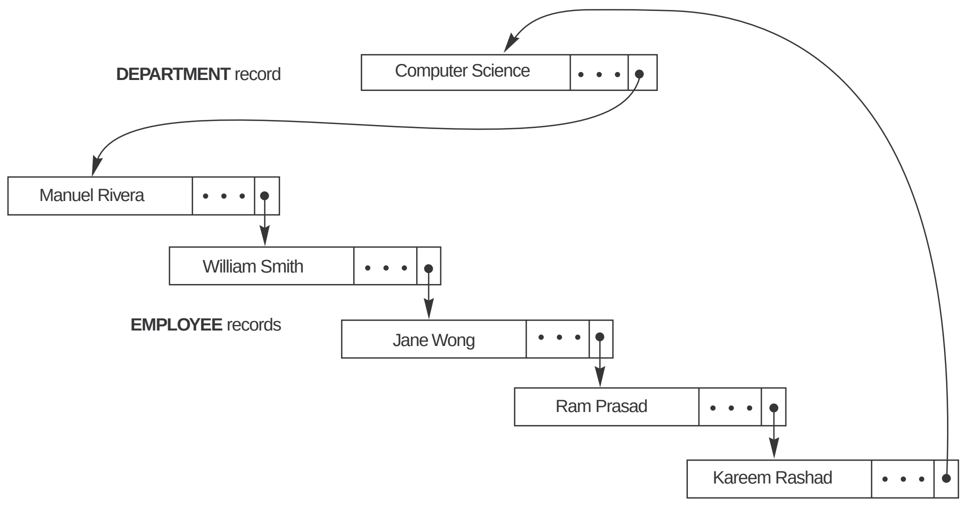


Figure E.4 Alternate representation of a set instance as a linked list.

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.

E.1.3 Special Types of Sets

One special type of set in the CODASYL network model is worth mentioning: SYSTEM-owned sets.

System-owned (Singular) Sets. A **system-owned** set is a set with no owner record type; instead, the system is the owner.[2] 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.

2. By *system*, we mean the DBMS software.

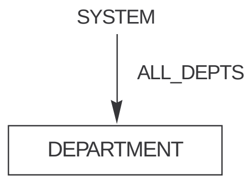


Figure E.5 A singular (SYSTEM-owned) set ALL_DEPTS.

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 E.4.2. 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 E.5, which allows DEPARTMENT records to be accessed in order of some field—say, NAME—with an appropriate set-ordering specification. Other special set types include recursive set types, with the same record serving as an owner and a member, which are mostly disallowed; multimember sets containing multiple record types as members in the same set type are allowed in some systems.

E.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 E.4. 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 E.4, 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, ith, or last member record of the set occurrence. If no such record exists, return an exception code.
- Given a member record, find the next (or previous) member record of the set occurrence. If no such record exists, return an exception code.
- 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, which means that a record type may participate as owner and member in numerous set types. For example, in the network schema that appears later as Figure E.8, 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:* For each set type an additional **OWNER** pointer is included in the member record type that 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 ith element in the array points to the ith 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).

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.

E.1.5 Using Sets to Represent 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, the extra 1:1 constraint must be imposed by the application program.

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 E.6(a) nor that in Figure E.6(b) will represent the relationship correctly. Figure E.6(a) enforces the incorrect constraint that a PROJECT record is related to only one EMPLOYEE record, whereas Figure E.6(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 E.6(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.

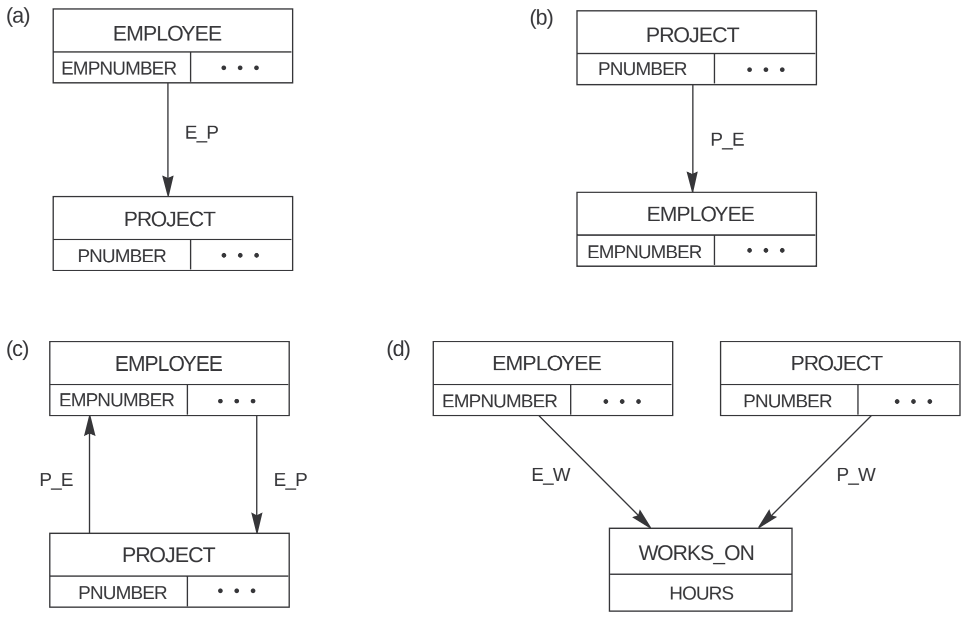
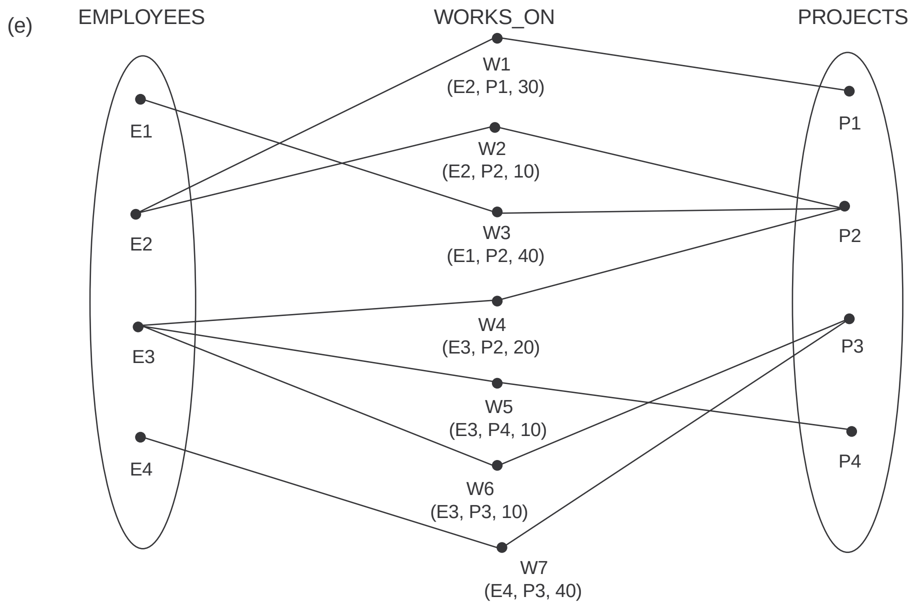


Figure E.6 Representing M:N relationships. (a)–(c) Incorrect representations. (d) Correct representation using a linking record type.



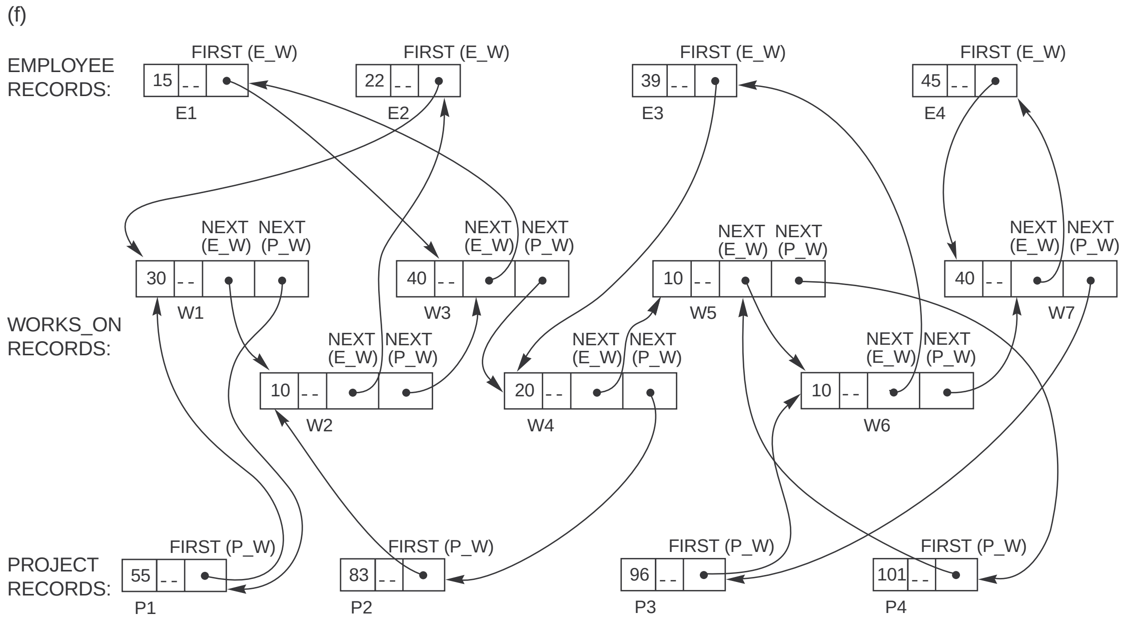


Figure E.6 *(Continued)* (e) Some instances. (f) Using linked representation.

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 E.6(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 E.6(e).

Figure E.6(f) shows an example of individual record and set occurrences in the linked list representation corresponding to the schema in Figure E.6(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 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 E.6(f) are shown in Figure E.6(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 E.4.2.

Notice that we could represent the M:N relationship as in Figure E.6(a) or (b) if we were allowed to duplicate PROJECT (or EMPLOYEE) records. In Figure E.6(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.

E.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 E.3). Not all combinations of the constraints are possible. We first discuss each type of constraint and then give the allowable combinations.

E.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 E.4.3). There are two options:

- AUTOMATIC: The new member record is *automatically connected* to an appropriate set occurrence when the record is inserted.[3]
- 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 E.2. 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 E.4.4.

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 E.7(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.

E.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

3. The appropriate set occurrence is determined by a specification that is part of the definition of the set type, the SET OCCURRENCE SELECTION.

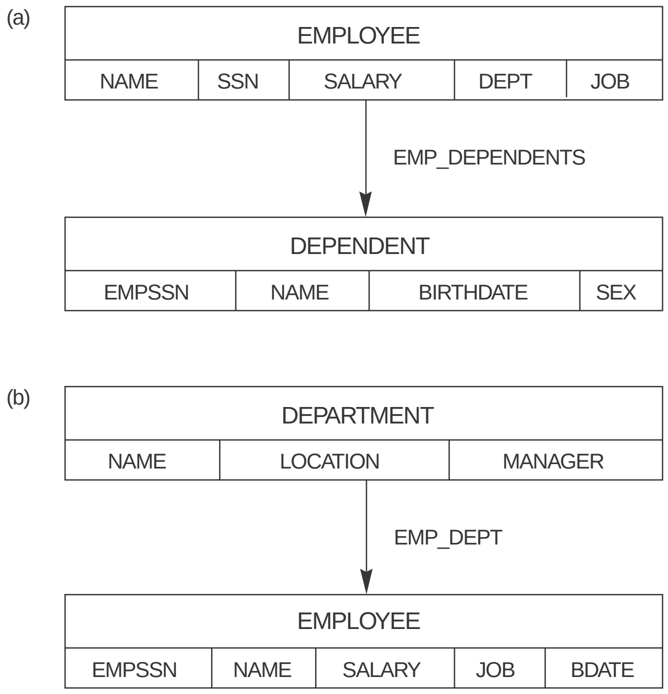


Figure E.7 Different set options. (a) An AUTOMATIC FIXED set. (b) An AUTOMATIC MANDATORY set.

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 will by means of the CONNECT and DISCONNECT commands of the network DML (see Section E.4.4).
- 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 E.4.4).
- 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 E.2. 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

Table E.1 Set Insertion and Retention Options

	Retention Option		
	OPTIONAL	**MANDATORY**	**FIXED**
MANUAL	Application program is in charge of inserting member record into set occurrence. Can CONNECT, DISCONNECT, RECONNECT	Not very useful.	Not very useful.
AUTOMATIC	DBMS inserts a new member record into a set occurrence automatically. Can CONNECT, DISCONNECT, RECONNECT.	DBMS inserts a new member record into a set occurrence automatically. Can RECONNECT member to a different owner.	DBMS inserts a new member record into a set occurrence automatically. *Cannot* RECONNECT member to a different owner.

E.7(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 E.7(b) a MANDATORY set EMP_DEPT relates an employee to the department the 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.

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. Table E.1 summarizes the Insertion and Retention options.

E.2.3 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.

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 E.3 in connection with the network model data definition language (DDL).

E.2.4 Set Selection Options

The following options can be used to select an appropriate set occurrence:

- 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 Figures E.8 and E.9(b).
- SET SELECTION BY APPLICATION: The set occurrence is determined via the application program, which should make the desired set occurrence the "current of set" before the new member record is stored. The new member record is then automatically connected to the current set occurrence.

E.2.5 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. Network DDL declarations for the record types of the COMPANY schema shown in Figure E.8 are shown in Figure E.9(a). Each record type is given a name by using the **RECORD NAME IS** clause. Figure E.9(b) shows network DDL declarations for the set types of the COMPANY schema shown in Figure E.8. 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 E.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.

E.3 Data Manipulation in 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 (DML)** is used for these purposes. The DML

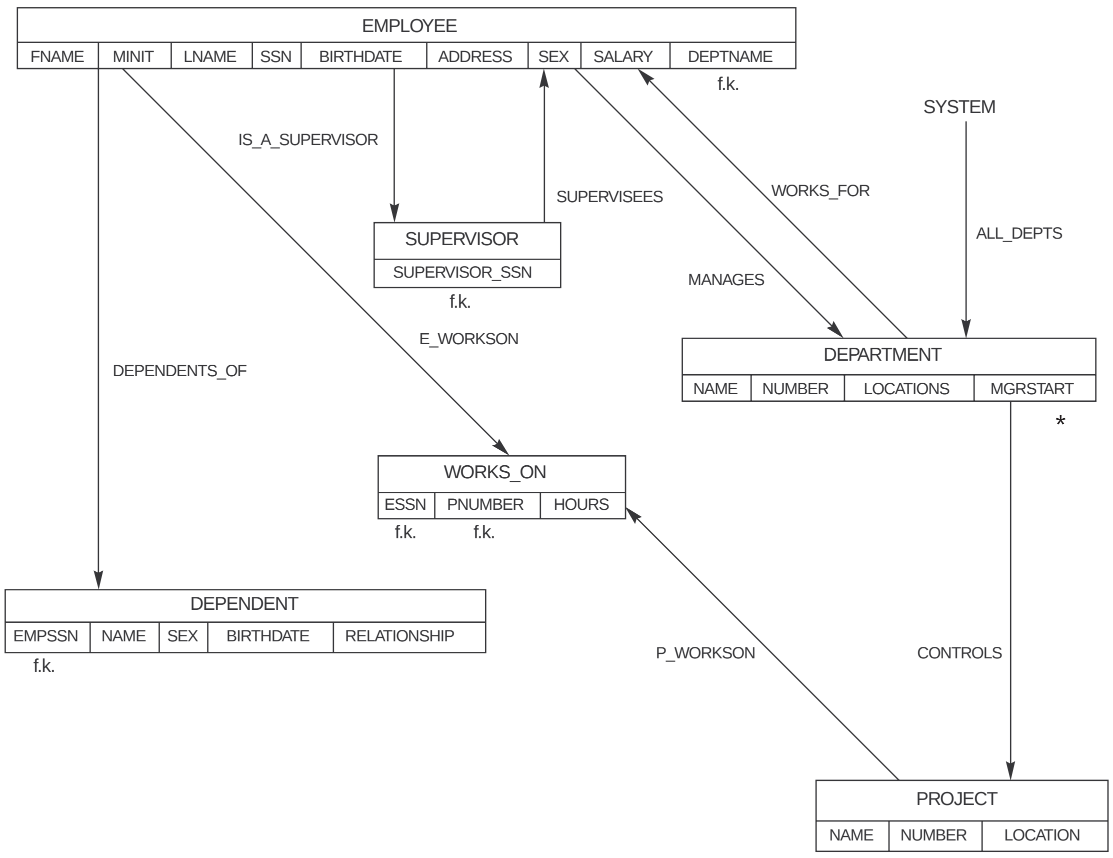


Figure E.8 A network schema diagram for the COMPANY database.

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.** Embedded commands of the DML are also called the **data sublanguage.** In practice, the most commonly used host languages are COBOL[4] and PL/I. In our examples, however, we show program segments in PASCAL notation augmented with network DML commands.

E.3.1 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 data-

4. The CODASYL DML in the DBTG report was originally proposed as a data sublanguage for COBOL.

```
(a)  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 E.9(a) Network DDL. (a) Record type declarations.

```
(b)  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 E.9(b) Network DDL. (b) Set type declarations.

```
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 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 E.9(b) *(Continued)*

base 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)**. 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.

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.
- *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.

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.

E.4 Network Data Manipulation Language

The commands for manipulating a network database are called the network **data manipulation language (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 E.2.

Table E.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.

We discuss these DML commands below and illustrate our discussion with examples that use the network schema shown in Figure E.8 and defined by the DDL statements in Figures E.9(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 in the actual DBMSs in use, COBOL and FORTRAN are predominantly used as host languages. The examples consist of short program segments without any variable declarations. In our programs, we prefix the DML commands with a $-sign to distinguish them from the PASCAL language statements. We write PASCAL language keywords—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.

E.4.1 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 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   EMPLOYEE.FNAME := 'John'; EMPLOYEE.LNAME := 'Smith';
     2   $FIND ANY EMPLOYEE USING FNAME, LNAME;
     3   if DB_STATUS = 0
     4      then begin
     5         $GET EMPLOYEE;
     6         writeln (EMPLOYEE.SALARY)
     7         end
     8      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.” The IDMS system combines FIND and GET into a single command, called OBTAIN.

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.

E.4.2 DML Commands for Set Processing

For set processing, we have the following variations of FIND:

- **FIND** (**FIRST** | **NEXT** | **PRIOR** | **LAST** | ...) <record type name>
- **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.

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 EX3:

```
EX3: 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 EX3, 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.

We can 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.

E.4.3 DML Commands for Updating the Database

The DML commands for updating a network database are summarized in Table E.2. Here, we first discuss the commands for updating records—namely the STORE, ERASE, and 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 CONNECT, DISCONNECT, and RECONNECT commands.

The STORE Command. The **STORE** command is used to insert a new record. Before issuing a STORE, we must first set up the UWA variable of the corresponding record type so that its field values contain the field values of the new record. For example, to insert a new EMPLOYEE record for John F. Smith, we can prepare the data in the UWA variables, then issue

```
$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.

The ERASE and ERASE ALL Commands. 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 above, we can use EX4:

```
EX4:  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. 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.

The MODIFY Command. The final command for updating records is the **MODIFY** command, which changes some of the field values of a record.

E.4.4 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 EX5:

```
EX5:  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;
```

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 command 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 EX6:

```
EX6:  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 MANDATORY sets, but not with FIXED sets. The RECONNECT command moves a member record 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.

Selected Bibliography

Early work on the network data model was done by Charles Bachman during the development of the first commercial DBMS, IDS (Integrated Data Store, 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.

The DBTG of CODASYL 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 (American National Standards Institute) proposed a standard network language called NDL.

The design of network databases is discussed by Dahl and Bubenko (1982), Whang et al. (1982), Schenk (1974), Gerritsen (1975), and Bubenko et al. (1976). Irani et al. (1979) discuss optimization techniques for designing network schemas from user requirements. Bradley (1978) proposes a high-level query language for the network model. Navathe (1980) discusses structural mapping of network schemas to relational schemas. Mark et al. (1992) discuss an approach to maintaining a network and relational database in a consistent state.

Other popular network model-based systems include VAX-DBMS (of Digital), IMAGE (of Hewlett-Packard), DMS-1100 of UNIVAC, and SUPRA (of Cincom).