Appendix L

The Network Database Model

Preview

In this appendix, you will learn about network database model implementation. (You learned about the network database model concepts in Chapter 2, Data Models.) Like the hierarchical database model, the network model may be represented by a tree structure in which 1:M relationships are maintained. However, the network model easily handles complex multiparent relationships without resorting to the creation of logical (as opposed to physical) database links.

Also remember from Chapter 2 that a close kinship exists between hierarchical and network models. For example, the network model's owner corresponds to the hierarchical model's parent, and the network model's member corresponds to the hierarchical model's child. However, the network model places a set between the owner and the member, using that set to describe the relationship between the two. You will see that the set makes it possible to describe more complex relationships between owners and members than was feasible in the hierarchical model.

Although the pointer movement is more complex in the network model than in its hierarchical counterpart, the DBMS creates and maintains the pointer system, making it transparent to the user and even to the applications programmer. However, the cost of such pointer system transparency is greater system complexity. For example, you will learn that the schema requires careful delineation of the model's components. In short, the network model requires the database administrator to pay close attention to the model's physical environment. In turn, application programmers must note the model's physical details.

Data Files and Available Formats

MS Access | Oracle | MS SQL | My SQL

MS Access | Oracle | MS SQL | My SQL

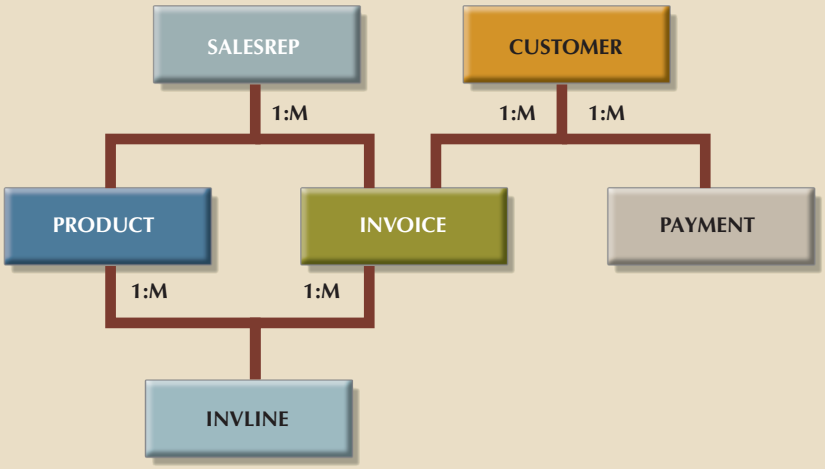
There are no data files for this appendix.

Data Files Available on cengagebrain.com

L-1 A Quick Review of Basic Network Data Model Concepts

In Chapter 2, you learned that the network model's end users *perceive* the network database to be a collection of records in one-to-many (1:M) relationships. But unlike the hierarchical database model, a record in the network database model can have more than one parent. Both 1:M and multiparent relationships are evident in the ROBCOR database shown in Figure L.1. Figure L.1 depicts a simple network database model for the ROBCOR Corporation. ROBCOR engages in retail sales, and its management wants to automate both the sales and the billing operations.

FIGURE L.1 A NETWORK MODEL OF THE ROBCOR DATABASE



The following basic network model concepts are illustrated in Figure L.1:

- SALESREP, CUSTOMER, PRODUCT, INVOICE, PAYMENT, and INVLINE represent record types.
- INVOICE is owned by both SALESREP and CUSTOMER. Similarly, INVLINE has two owners, PRODUCT and INVOICE.
- The network database model may still include *hierarchical* one-owner relationships (for example, CUSTOMER owns PAYMENT).

Finally, relationships among records must be decomposed into a series of *sets* before a network database model can be implemented. For example, Figure L.2 shows two sets that describe the relationships between owners and members.

1. The SALES set includes all of the INVOICE tickets that belong to a specific CUSTOMER.
2. The PAIDBY set defines the relationship between CUSTOMER (the owner of the set) and PAYMENT (the member of the set).

ROBCOR's network database model contains other sets, too. In fact, before the network database model can be implemented, *all* of its data structures must be decomposed into sets of 1:M relationships. The sets that can be defined for Figure L.2's network database model are listed in Table L.1. Each set listed represents a relationship between owners and members. When you implement a network database design, every set must be given a name and all owner and member record types must be defined.

FIGURE L.2 SET ILLUSTRATION

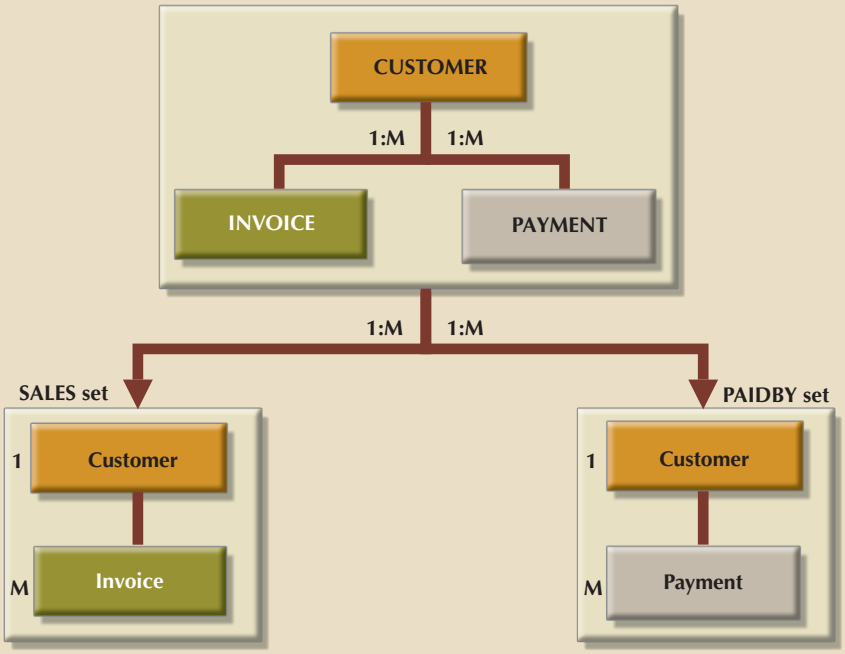


TABLE L.1

A TABLE OF SETS FOR THE NETWORK MODEL SHOWN IN FIGURE L.2

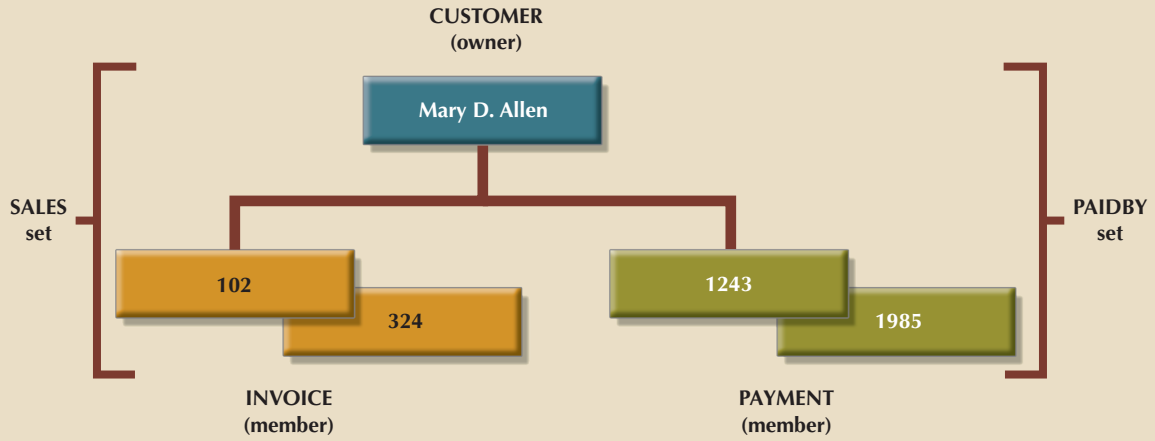
SET NAME	OWNER	MEMBER
PAIDBY	CUSTOMER	PAYMENT
SALES	CUSTOMER	INVOICE
INVLINE	INVOICE	INVLINE
COMMISSION	SALESREP	INVOICE
PRODSOLD	PRODUCT	INVLINE

Figure L.3 depicts the contents of the member records for the PAIDBY set and the SALES set. The Mary D. Allen owner record has been used to illustrate how the data fit into the network structure. As you examine Figure L.3, note that the CUSTOMER named Mary D. Allen is the owner of two sets, SALES and PAIDBY. Mary D. Allen is also the owner of two INVOICE records, Invoices 102 and 324, which are members of the SALES set. She is also the owner of two PAYMENT records, Payments 1243 and 1985, which are members of the PAIDBY set. Given that structure, the user may navigate through any one of those two sets, using the data manipulation language (DML) provided by the DBMS.

Keep in mind that the INVOICE and PAYMENT records shown in Figure L.3 are related only to the CUSTOMER Mary D. Allen. When the user accesses another CUSTOMER record, a different series of INVOICE and PAYMENT records are available for that customer. Therefore, network database designers must also be aware of currency. The word **currency** indicates the position of the record pointer within the database and *refers to the most recently accessed record.*

currency
This term indicates the position of the record pointer within the database and *refers to the most recently accessed record.*

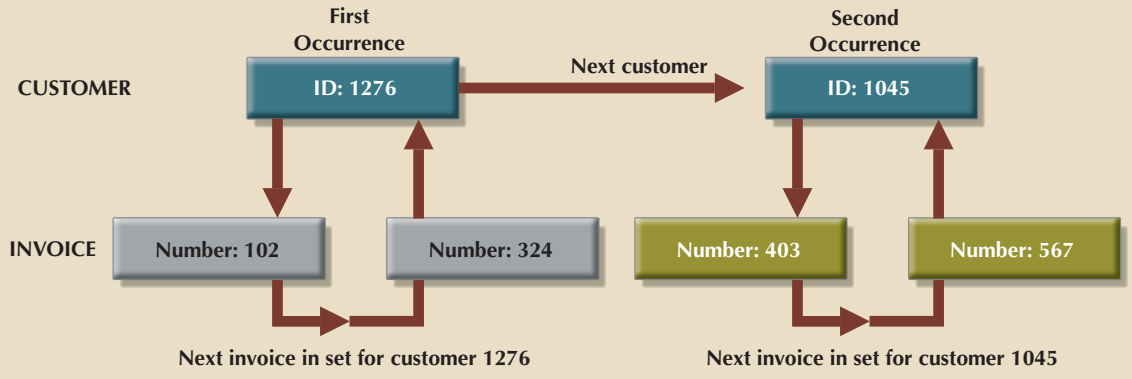
FIGURE L.3 CONTENTS OF SALES AND PAIDBY SETS



The DBMS automatically updates the pointers after the execution of each operation. A pointer exists for each record type in a set. A pointer's current value refers to a current record. Actually, two types of currency pointers exist: a record pointer and a set pointer. Record pointers, also known as record type pointers, exist for each record type within the database, *and they always point to the current record within each record type*. Because a set must contain two record types, an owner and a member, the set pointer points to either an owner record or a member record.

To illustrate the use of pointers, let's examine the condition shown in Figure L.4. Figure L.4 depicts two occurrences of the SALES set. The first occurrence corresponds to CUSTOMER number 1276. The second occurrence corresponds to CUSTOMER number 1045. Note that the occurrences are determined by the owner of the set: Every time you move to a new CUSTOMER record, a new group of INVOICE member records is made available.

FIGURE L.4 TWO OCCURRENCES WITHIN THE SALES SET



Given the SALES set components, the owner record is CUSTOMER and the member record is INVOICE. Therefore, the pointer locations for the current record of each record type and for the current record of a set *after the completion of each operation* will correspond to those found in Table L.2.

TABLE L.2

POINTER LOCATIONS WITHIN THE SALES SET

	RECORD TYPE POINTERS		CURRENT RECORD OF THE SET
OPERATION	CUSTOMER POINTER	INVOICE POINTER	SET POINTER
LOCATE CUSTOMER=1276 [a]	1276	NULL [b]	CUSTOMER RECORD (1276)
LOCATE FIRST IN SALES SET [c]	1276	102	INVOICE RECORD (102)
LOCATE NEXT IN SALES SET [d]	1276	324	INVOICE RECORD (324)
LOCATE INVOICE=403	1276	403	INVOICE RECORD (403)
LOCATE OWNER IN SALES SET	1045	403	CUSTOMER RECORD (1045)

[a] The summary in this table employs a pseudosyntax. For example, LOCATE CUSTOMER=1276 indicates a search for a CUSTOMER record whose customer number is 1276.
[b] Using network database jargon, NULL is used to indicate that no operation has been initiated yet and no pointer location has yet been designated.
[c] LOCATE FIRST IN SALES SET means "Locate the *first member record in the current SALES set*."
[d] LOCATE NEXT IN SALES SET means "Locate the *next member record in the SALES set*."

L-2 The Database Definition Language (DDL)

The database standardization efforts of the Data Base Task Group (DBTG) led to the development of standard **data definition language (DDL)** specifications. Those specifications include DDL instructions that are used to define a network database.

The examples of DDL will be based on Honeywell's Integrated Data Store/II (L-D-S/II) network database management system. Since L-D-S/II's DDL is very extensive, a subset of it will be used. (If you want to learn more about L-D-S/II, look at an L-D-S/II reference manual.) Don't despair if you don't have access to L-D-S/II; because the network model is based on CODASYL standards, database definition and creation are similar when other commercial applications are used.

Network database definition and creation are not interactive processes. Therefore, they must be done through the use of DBMS utility programs at the system prompt level. Creating an L-D-S/II database requires three steps: a logical definition of the database, using the DDL; a physical definition of the database, using the DMCL (device media control language); and the physical creation of the database storage files on secondary storage.

To see what general procedures are followed to design, create, and manipulate a network database, examine Figure L.5. (The illustration is based on CP-6 L-D-S/IL. CP-6 is a Honeywell operating system.)

The network database schema view or **schema** describes the entire database as seen by the database administrator. The schema defines the database name; the record type for each record; and the field, set, owner, and member records. The database subschema view, or *subschema*, describes the portion of the database used by each application program.

As you examine Figure L.5, note that the schema view and subschema view(s) are normal text files. Those schema and subschema text files may be created with any text-processor program. The files contain DDL and DMCL instructions, which describe the database and application views of the database and indicate what utility programs must be invoked to validate and create the database structure. Subschema views must be defined and validated for each application that uses the database.

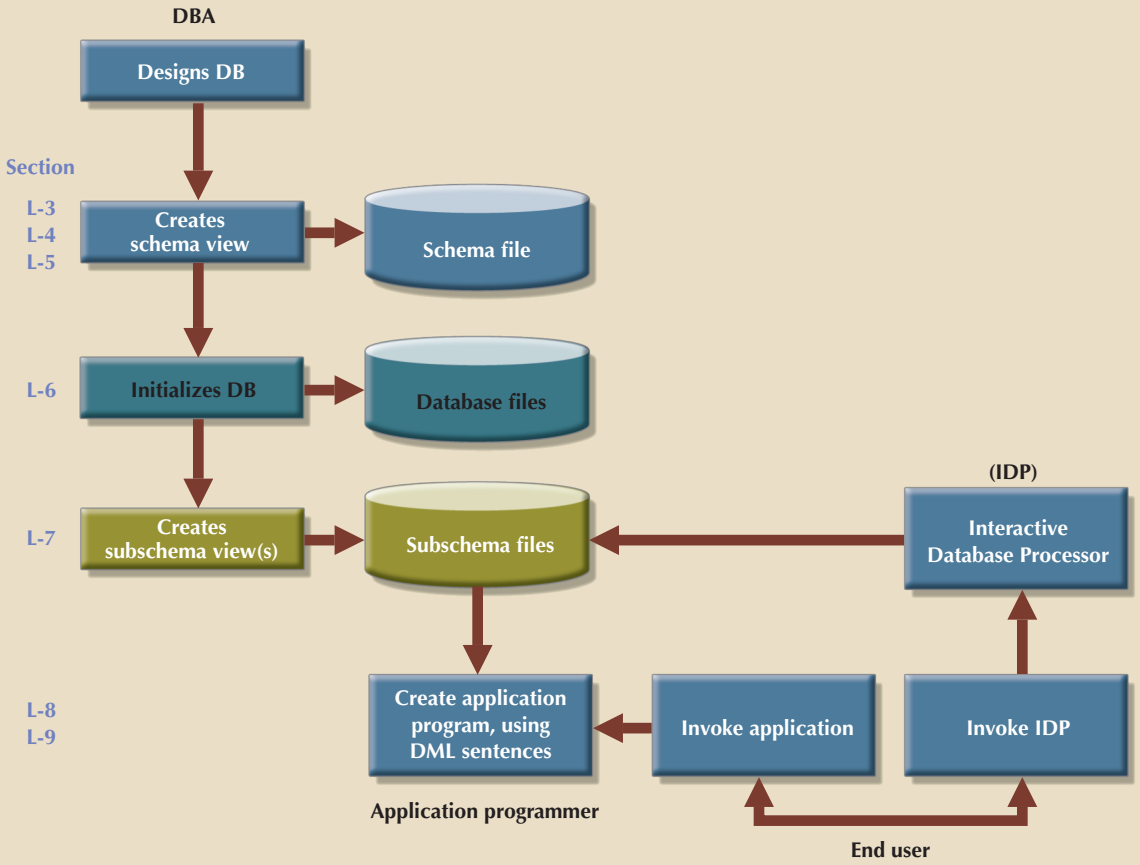
L-D-S/II has an **Interactive Database Processor (IDP)**, which allows users to manipulate databases. The IDP front end is intended for users who have some programming knowledge; it is not well suited for most end users.

data definition language (DDL)
The language that allows a database administrator to define the database structure, schema, and subschema.

schema
A logical grouping of database objects, such as tables, indexes, views, and queries, that are related to each other. Usually, a schema belongs to a single user or application.

Interactive Database Processor (IDP)
In an IDS/II network DBMS, a processor that allows users to manipulate databases. The IDP front end is intended for users who have some programming knowledge and is not well-suited for most end users.

FIGURE L.5 A FLOW DIAGRAM FOR THE CREATION OF A NETWORK DATABASE



L-3 The Schema Definition

The first step in implementing a network database is defining the entire database *as seen by the database administrator* (DBA). The ROBCOR database will be used to illustrate the schema definition. The complete DDL description for the ROBCOR database is listed in Table L.3.

TABLE L.3

SCHEMA DEFINITION FOR THE ROBCOR DATABASE

LINE NUMBER	DLL CODE
1	DBACS TRANSLATE SCHEMA ROBCOR DDL END
2	SCHEMA NAME IS ROBCOR
3	AREA NAME IS MTSU
4	RECORD NAME IS CUSTOMER
5	LOCATION MODE IS CALC USING CUSID
6	DUPLICATES ARE NOT ALLOWED
7	WITHIN MTSU
8	02 CUSID TYPE IS CHARACTER 5
9	02 CUSTNAME TYPE IS CHARACTER 20

TABLE L.3

SCHEMA DEFINITION FOR THE ROBCOR DATABASE (CONTINUED)

LINE NUMBER	DLL CODE
10	02 CUSTADDRESS TYPE IS CHARACTER 35
11	RECORD NAME IS INVOICE
12	LOCATION MODE IS CALC USING INVNUM
13	DUPLICATES ARE NOT ALLOWED
14	WITHIN MTSU
15	02 INVNUM TYPE IS DECIMAL 5
16	02 INVDATE TYPE IS DECIMAL 8
17	02 INVAMOUNT TYPE IS DECIMAL 6,2
18	RECORD NAME IS INVLINE
19	LOCATION MODE IS VIA INVLINE SET
20	WITHIN MTSU
21	02 LIPRO TYPE IS CHARACTER 4
22	02 LIQTY TYPE IS DECIMAL 2
23	02 LIPRICE TYPE IS DECIMAL 4,2
24	RECORD NAME IS PAYMENT
25	LOCATION MODE IS CALC USING PAYNUM
26	SET DUPLICATES ARE NOT ALLOWED
27	WITHIN MTSU
28	02 PAYNUM TYPE IS DECIMAL 5
29	02 PAYDATE TYPE IS DECIMAL 8
30	02 PAYAMOUNT TYPE IS DECIMAL 6,2
31	RECORD NAME IS SALESREP
32	LOCATION MODE IS CALC USING SLSNUM
33	SET DUPLICATES ARE NOT ALLOWED
34	WITHIN MTSU
35	02 SLSNUM TYPE IS DECIMAL 5
36	02 SLSDATE TYPE IS DECIMAL 8
37	RECORD NAME IS PRODUCT
38	LOCATION MODE IS CALC USING PRODNUM
39	SET DUPLICATES ARE NOT ALLOWED
40	WITHIN MTSU
41	02 PRODNUM TYPE IS CHARACTER 4
42	02 PRODDATE TYPE IS DECIMAL 8
43	02 PRODQTY TYPE IS NUMERIC 6,2
44	SET NAME IS SALES
45	OWNER IS CUSTOMER
46	SET IS PRIOR PROCESSABLE
47	ORDER IS PERMANENT
48	INSERTION IS NEXT
49	MEMBER IS INVOICE
50	INSERTION IS AUTOMATIC
51	RETENTION IS MANDATORY
52	LINKED TO OWNER
53	SET SELECTION IS THRU SALES
54	OWNER IDENTIFIED BY APPLICATION
55	SET NAME IS PAIDBY

TABLE L.3

SCHEMA DEFINITION FOR THE ROBCOR DATABASE (CONTINUED)

LINE NUMBER	DLL CODE
56	OWNER IS CUSTOMER
57	SET IS PRIOR PROCESSABLE
58	ORDER IS PERMANENT
59	INSERTION IS NEXT
60	MEMBER IS PAYMENT
61	INSERTION IS AUTOMATIC
62	RETENTION IS MANDATORY
63	LINKED TO OWNER
64	SET SELECTION IS THRU PAIDBY
65	OWNER IDENTIFIED BY APPLICATION
66	SET NAME IS INVLINE
67	OWNER IS INVOICE
68	SET PRIOR PROCESSABLE
69	ORDER IS PERMANENT
70	INSERTION IS NEXT
71	MEMBER IS INVLINE
72	INSERTION IS AUTOMATIC
73	RETENTION IS MANDATORY
74	LINKED TO OWNER
75	SET SELECTION THRU INVLINE
76	OWNER IDENTIFIED BY APPLICATION
77	SET NAME IS COMMISSION
78	OWNER IS SALESREP
79	SET PRIOR PROCESSABLE
80	ORDER IS PERMANENT
81	INSERTION IS NEXT
82	MEMBER IS INVOICE
83	INSERTION IS AUTOMATIC
84	RETENTION IS MANDATORY
85	LINKED TO OWNER
86	SET SELECTION IS THRU COMMISSION
87	OWNER IDENTIFIED BY APPLICATION
88	SET NAME IS PRODSOLD
89	OWNER IS PRODUCT
90	SET PRIOR PROCESSABLE
91	ORDER IS PERMANENT
92	INSERTION IS NEXT
93	MEMBER IS INVLINE
94	INSERTION IS AUTOMATIC
95	RETENTION IS MANDATORY
96	LINKED TO OWNER
97	SET SELECTION IS THRU PRODSOLD
98	OWNER IDENTIFIED BY APPLICATION
99	END SCHEMA

To understand the DDL sequence shown in Table L.3, you must first learn how the database components work together. Keep in mind that a network database is basically a system driven by pointers. Think of a network database as a system having two components: the data and the pointer structures. The data (records with fields) are the raw facts kept in permanent storage devices for processing and retrieval. The pointers represent the structure that models the data and describes the required relationships (sets).

More precisely, the pointers define the way relationships are represented among entities. When an application program stores data in the network database, two different structures are updated: the data and the sets (pointers). Remembering that information will help you understand the DDL commands used to describe the database.

L-4 An Explanation of the Schema Definition

Table L.3's main schema definition components may be described this way:

- Line 1 invokes the **Database Administrator Control System (DBACS)**. The DBACS is the database definition processor that reads the ROBCOR database definition and validates the schema. (The DBACS works like a compiler.)
- Line 2 defines the schema name, which may be up to 30 characters long. Line 3 invokes the *AREA* clause. An **AREA** is a section of physical storage space that is reserved to store the database. The AREA clause allows the (physical) storage of a database in more than one location, thereby improving access speed. The area name must be unique, and there must be at least one area defined for the database.

L-4a Record Definitions

In L-D-S/II, you must first define all of the required record types. Table L.3 illustrates how that is done.

- Lines 4–10 in Table L.3 yield the CUSTOMER record definition. The **RECORD NAME clause** initiates the record's definition by assigning it a unique name. A valid schema must contain at least one record type.
- The **LOCATION MODE clause** in Table L.3 determines where the record will be (physically) stored in the database and how the record will be retrieved. Four location modes are supported under L-D-S/II: DIRECT, CALC, VIA SET, and INDEXED.

  1. DIRECT is the fastest location mode and requires that the application program assign a unique numeric key for each record. Using the DIRECT approach, the user exercises direct control over the arrangement of the records in the database. The DIRECT location mode allows the application program (programmer) to exercise the greatest degree of control over the location and retrieval of records from the database.
  2. CALC mode in Table L.3 uses a hashing algorithm over a record's field to generate the database key for each record. The same algorithm is used to retrieve the records. The DBA indicates the field over which L-D-S/II will apply the algorithm in the schema definition, text. This method yields a uniform dispersion of the records across the database. Note the use of CALC in lines 5 and 6 of the database schema definition and observe that line 6 specifies that DUPLICATES of key values are not allowed for this record type.
  3. VIA … SET places member records together near the owner record occurrence in a set. The VIA … SET approach is particularly useful when member records will be accessed sequentially. Note especially line 19 in Table L.3's database schema

Database Administrator Control System (DBACS)
In the network model, the database definition processor that reads the database definition and validates the schema (The DBACS works like a compiler.).

AREA
A named section of permanent storage space that is reserved to store the database.

RECORD NAME clause
In the IDS/II network DBMS, this clause initiates the record's definition by assigning it a unique name. An IDS/II network database must contain at least one record type.

LOCATION MODE clause
In an IDS/II network DBMS, this clause determines where a record will be (physically) stored in the database and how the record will be retrieved.

definition: To store an INVLINE occurrence, the INVOICE record must first be stored; then the INVLINE records will be stored around it.

4. INDEXED defines an independent storage structure. Indexed records do not participate in any sets. Instead, indexes are stored in an independent file. A unique primary index is created over a record's field. The index represents the order in which the records are stored in the database. There can be one primary key and several secondary keys for each record. Note the example in Table L.4.

TABLE L.4

DEFINING THE INDEX FILE

LINE	CODE	COMMENTS
1	RECORD NAME IS JOB_HISTORY	
2	LOCATION MODE IS INDEXED USING EMP_NUM	Record field is EMP_NUM. Primary key is EMP_NUM.
3	WITHIN EMP_HISTORY	Area name is EMP_HISTORY.
4	KEY NAME IS EMP_NUM	
5	ASCENDING EMP_SSNUM	
6	DUPLICATES ARE NOT ALLOWED	
7	KEY NAME IS JOB_DATE	Secondary key is JOB_DATE.
8	ASCENDING DATE_EMP	Record field is DATE_EMP.
9	DUPLICATES ARE NOT ALLOWED	
10	02 EMP_NUM TYPE IS DECIMAL 9	Record field
11	02 EMP_NAME TYPE IS CHARACTER	Record field
	02 DATE_EMP TYPE IS DECIMAL 8	Record field
	02 COMP_NAME TYPE IS CHARACTER	Record field
	02 JOB_SALARY TYPE IS DECIMAL 6	Record field

As you examine Table L.4, note these features:

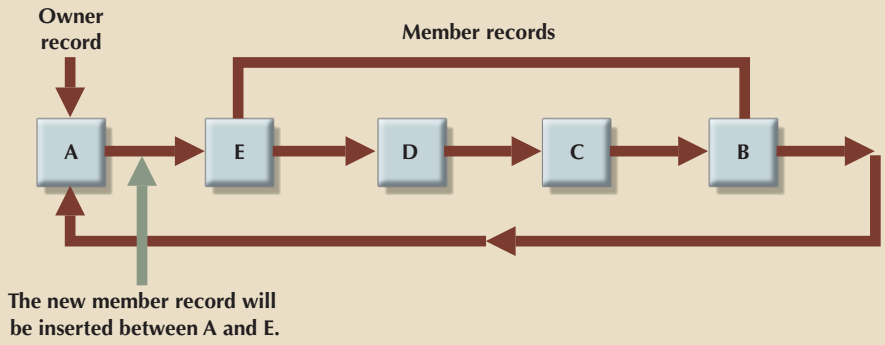
- EMP_NUM represents the record's key field.
- There can be only one record type in an indexed area.
- Secondary keys may also be defined for the record.
- The DUPLICATES clauses determine whether the system will allow the use of duplicate primary key values. Because each CUSTOMER must have a unique customer identification number (CUSID) in the application, the DUPLICATES clause specifies that duplicates are not allowed.
- The WITHIN clauses specify in which area the records will be stored. In this case, all of the records that make up the ROBCOR database will be stored in the area named MTSU.
- The TYPE clauses allow you to define any of the following data types: fixed binary, float hexadecimal, decimal, character, database key, or unspecified (string). The database key will store the record key, and the unspecified data type is a string. (Note that field definitions in L-D-S/II resemble COBOL data definitions.) Using a COBOL-like syntax, you may define the name of the field, the TYPE of the field, and its length.

L-4b Set Definitions

After you have defined all of the records that make up your database, you must define the sets or relations among record types. Lines 44–54 in Table L.3 yield a set definition, as follows:

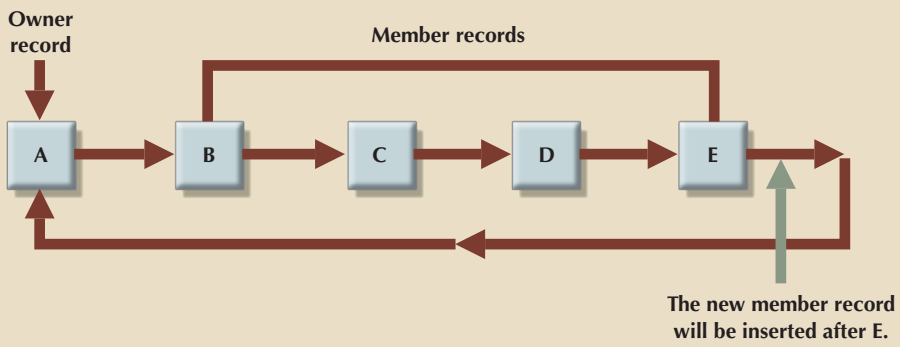
- Line 44 names the set. Note that the name must be unique within the current schema.
- Line 45 identifies the OWNER record type, which must be a valid record type already defined in the schema. A record can be an owner or a member of more than one set. However, a set may have only one OWNER. Line 46's SET IS PRIOR PROCESSABLE clause allows the L-D-S/II system to include a pointer to the previous record, thereby allowing efficient backward processing.
- The ORDER clause (lines 47 and 48) specifies where the record will be inserted within the set. The INSERTION clause can be FIRST, LAST, NEXT, or PRIOR.
- FIRST and LAST refer to the owner record. Specifically, FIRST defines the position directly *after* the owner record. The use of FIRST yields a chronologically reverse-ordered set, as shown in Figure L.6.

FIGURE L.6 CHRONOLOGICALLY REVERSE-ORDERED SET



- LAST defines the position directly *before* the owner record in the set, yielding a chronologically ordered set shown in Figure L.7.

FIGURE L.7 CHRONOLOGICALLY ORDERED SET



- NEXT and PRIOR in Table L.3 refer to the position relative to the current record of the set. The current record may be either the owner or a member of the set, whichever was last selected.
- The MEMBER clause in Table L.3 identifies the set's member record type. (The record must have been defined previously.) A given set may contain several member record types.
- The INSERTION and RETENTION clauses in Table L.3 define the way in which L-D-S/II associates the member records with their respective owner records. Valid parameters are shown in Table L.5.

TABLE L.5

VALID INSERTION AND RETENTION PARAMETERS

CODE	COMMENT
INSERTION IS {AUTOMATIC}	The member record is automatically a member of the set when it is first stored.
INSERTION IS {MANUAL}	The record is not a member of any set when it is first stored. It can be related (manually) to a set later.
RETENTION IS {MANDATORY}	A member record should always belong to a set.
RETENTION IS {OPTIONAL}	The member record does not need to belong to a set.

- The INSERTION clause in Table L.3 specifies when a member record will be linked to an owner record.
- The RETENTION clause indicates whether a record should always belong to a specified set (mandatory) or if it can be in the database without belonging to any set (optional). The definition of the INSERTION and RETENTION parameters helps assure enforcement of the database's integrity.
- The LINKED TO OWNER clause in Table L.3 creates a pointer to the member record's owner. Such a link allows you to find the owner when the current record is a member record.
- The SET SELECTION clause in Table L.3 (lines 53 and 54) identifies how the current record of a set is selected prior to the record's retrieval or insertion. The application program will identify the owner record first, then make that record the current record before allowing the insertion or retrieval of any member record.

You can see that the network model uses several pointers to create the database's logical structure. For example, the database schema includes pointers to the next record, pointers to the prior record, pointers to the owner record, and so on. The degree of physical detail involved in the database definition is also very clear. As a result, learning the intricacies of such a database environment takes a considerable amount of time and effort.

Network database programmers must also be familiar with the available storage structures at the physical level. The DBACS not only validates and translates the schema specifications, but also defines and validates the database's physical storage characteristics. (The physical characteristics are defined using a device media control language.)

L-4c Device Media Control Language

After defining the database schema, the database administrator (DBA) must define the physical storage characteristics. For example, the system must "know" how the database will be stored on disk, what the area name is, and what records and sets belong to

the specified area. The ROBCOR schema's DMCL is depicted in Table L.6. The DBACS translates the schema and DMCL files to validate the physical structure that will support the database.[1]

TABLE L.6

THE ROBCOR SCHEMA DMCL

LINE NUMBER	DLL CODE
1	DBACS TRANSLATE SCHEMA ROBCOR DCML
2	MODE IS ALTER
3	END
4	SCHEMA NAME IS ROBCOR
5	AREA NAME IS MTSU
6	ALLOCATE 512 DATA_BASE_KEYS
7	RECORD NAME IS CUSTOMER TYPE IS 1
8	RECORD NAME IS INVOICE TYPE IS 2
9	RECORD NAME IS PAYMENT TYPE IS 3
10	RECORD NAME IS INVL1NE TYPE IS 4
11	RECORD NAME IS SALESREP TYPE IS 5
12	RECORD NAME IS PRODUCT TYPE IS 6
13	SET NAME IS SALES
14	SET NAME IS PAIDBY
15	SET NAME IS INVLINE
16	SET NAME IS COMMISSION
17	SET NAME IS PRODSOLD
18	END_DMCL

The DMCL file contains the following five components:

1. The schema name.
2. The area name and physical characteristics.
3. The record definitions.
4. The set entry.
5. The key entries used to name all of the record keys found in the area.

L-5 Database Initialization

After the schema DDL and DMCL have been validated by the DBACS, the database must be initialized using a utility program named DBUTIL. The database initialization process creates the physical files that will contain the database. The database files will be located in the physical storage devices identified in the AREA clause specified in the DDL and DMCL schema files.

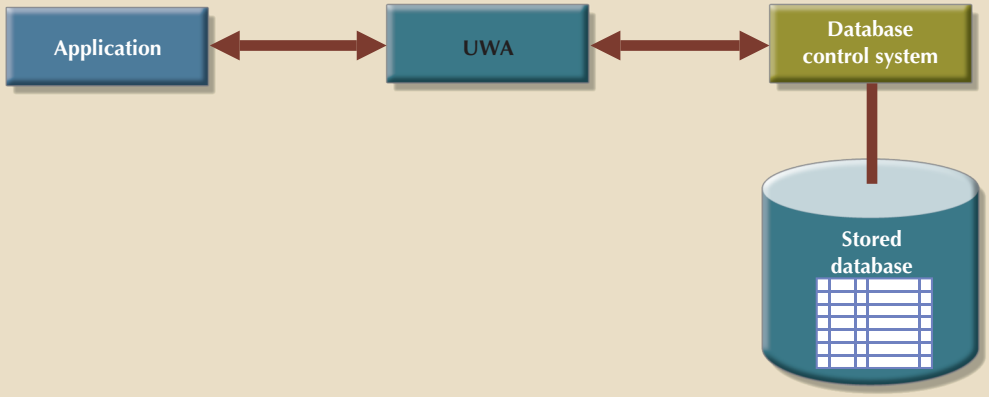
[1] A complete description of all possible system options is available in Honeywell's *CP-6 L-D-S/II Database Administrator Manual*, order number CE36-02.

L-6 Subschema Definition

All applications programs view L-D-S/II databases through a *subschema*. The **subschema** contains all of the fields, records, and sets that are available to the application. In effect, the subschema is a "window" that the DBCS (Data Base Control System) opens to the application. The application uses this window to communicate with the database. Keep in mind that the subschema is contained within the database's (total) conceptual schema. The DBCS validates all subschema entries against the schema. When an application program invokes a subschema, a User Work Area (UWA) is created by the DBCS. The **UWA** is a specific area of memory that contains several fields used to access and inform regarding the status of the database. The UWA also contains space for each record type defined in the subschema.

The UWA allows the application to communicate with the DBCS. Application programs read from and write to the UWA when the database is accessed or updated. Application programs can also check the database status after each operation to see if the operation was performed properly. The UWA's role as the interface between an application and the database is illustrated in Figure L.8.

FIGURE L.8 THE UWA AS AN INTERFACE BETWEEN THE APPLICATION AND DATABASE



subschema
In the network model, the portion of the database "seen" by the application programs that produce the desired information from the data in the database.

UWA
User Work Area; a specific area of memory that contains several fields used to access and inform regarding the status of the database. The UWA also contains space for each record type defined in the subschema.

When an application retrieves a database record, the DBCS reads that record and places it in the space reserved for it by the application program's UWA. The DBCS also updates all of the required UWA status fields. The application can also check and validate the database status after its last operation.

Subschemas are created manually by the DBA. In this case, the DBA must assure that all subschema definitions are correct and valid to the schema. A better way to create subschemas is to use the DBACS to create a full subschema from the main schema. That subschema will allow an unconstrained manipulation of the entire database. This all-encompassing subschema can then be modified by erasing all of the fields, records, and relations not required by the application program. L-D-S/II can generate subschemas for APL, BASIC, COBOL, and FORTRAN.

Table L.7 shows a COBOL subschema definition for the ROBCOR database. If this subschema is used, all sets and records of the database are available to the application program.

TABLE L.7

COBOL SUBSCHEMA DEFINITION FOR THE ROBCOR DATABASE

LINE NUMBER	DLL CODE
1	DBACS TRANSLATE COBOL SUBSCHEMA SUB_ROBCOR DDL END
2	TITLE DIVISION
3	SS SUB_ROBCOR WITHIN ROBCOR
4	MAPPING DIVISION
5	STRUCTURE DIVISION
6	REALM SECTION
7	RD MTSU
8	SET SECTION
9	SD COMMISSION
10	SD INVLINE
11	SD PAIDBY
12	SD PRODSOLD
13	SD SALES
14	KEY SECTION
15	RECORD SECTION
16	01 CUSTOMER
17	02 CUSTID DISPLAY PIC X(5)
18	02 CUSTNAME DISPLAY PIC X(20)
19	02 CUSTADDRESS DISPLAY PIC X(35)
20	01 INVLINE
21	02 LINEPROD DISPLAY PIC X(4)
22	02 LINEQTY DISPLAY PIC 9(2)V9(2)
23	02 LINEPRICE DISPLAY PIC 9(2)V9(2)
24	01 INVOICE
25	02 INVNUM DISPLAY PIC 9(5)
26	02 INVDATE DISPLAY PIC 9(8)
27	02 INVAMOUNT DISPLAY PIC 9(4)V9(2)
28	01 PAYMENT
29	02 PAYNUM DISPLAY PIC 9(5)
30	02 PAYDATE DISPLAY PIC 9(8)
31	02 PAYAMOUNT DISPLAY PIC 9(4)V9(2)
32	01 PRODUCT
33	02 PRODNUM DISPLAY PIC X(4)
34	02 PRODNAME DISPLAY PIC X(20)
35	02 PRODQTY DISPLAY PIC 9(4)V9(2)
36	01 SALESREP
37	02 SLSNUM DISPLAY PIC X(4)
38	02 SLSNAME DISPLAY PIC X(20)
39	END

Note that the subschema defined in Table L.7 contains the following three main components:

1. The *Title Division*, containing schema and subschema names.
2. The *Mapping Division*, containing all aliases used in the subschema.
3. The *Structure Division*, in which the area, sets, keys, and records used by the application are defined.

Given the components of the subschema depicted in Table L.7, note that a database file is referenced by its *realm*, rather than by its *area*, as is done in the schema definition. Actually, *realm* and *area* refer to the same thing. (The use of different terms to describe the same thing reflects the early lack of database terminology standards.)

The DBACS processes and validates the subschema definition in two independent steps. Those steps must be completed before any application programs based on the subschema can be compiled. (Keep in mind that the ROBCOR schema DDL and DMCL must be processed before the subschema can be processed.)

Figure L.9 shows the arrangement of the interactions between the DBMS and the ROBCOR information system components.

As you examine Figure L.9, note that each application program is given a UWA to manipulate the subset of the database needed by the application. The UWA is created at run time and uses the application's subschema definition data. Also remember that each subschema is a subset of the global database schema and must, therefore, be validated against the global schema. The DBMS (the DBCS component in L-D-S/II) is responsible for coordinating and controlling all of the interactions between the application programs, the user work areas, and the database.

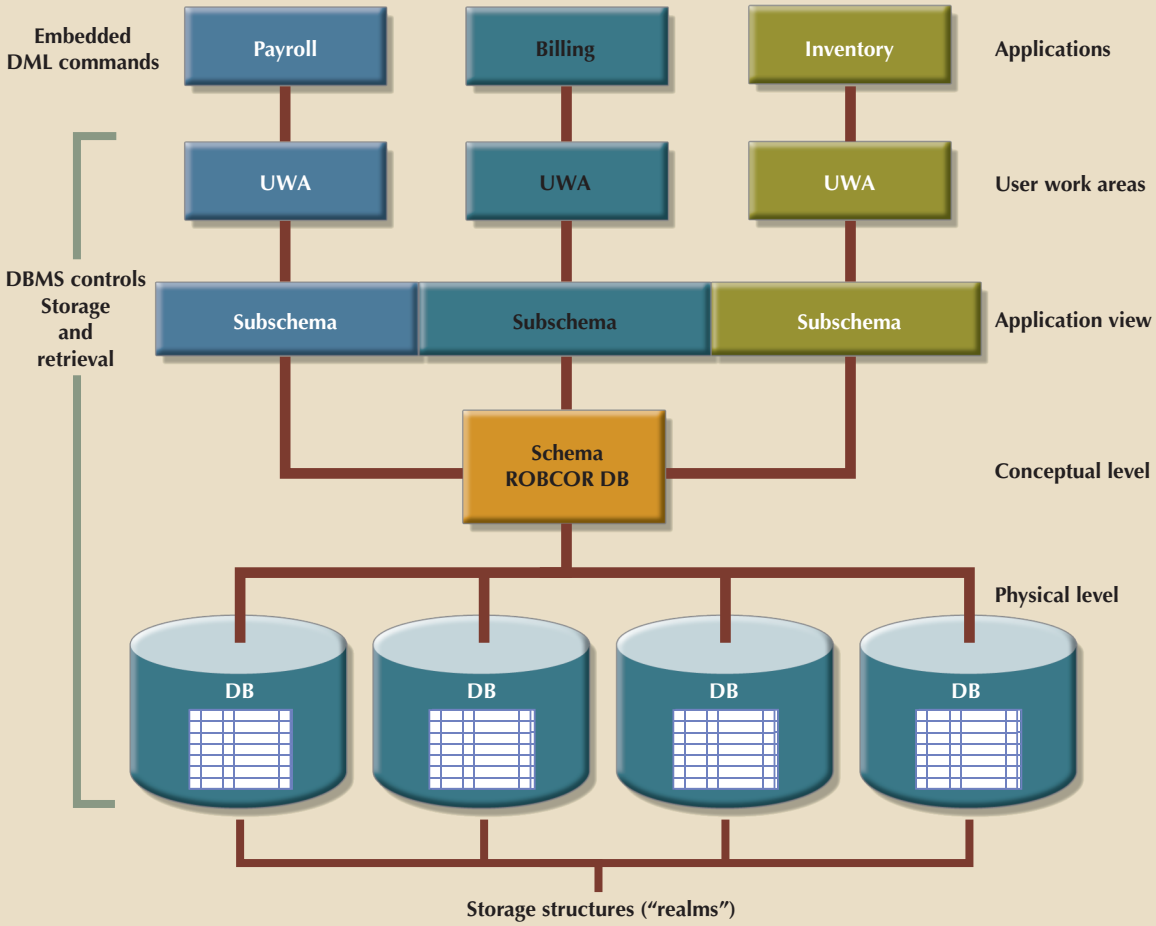
L-7 An Introduction to the Data Manipulation Language

Application programs navigate in a database by using a *data manipulation language (DML)*. As previously noted, L-D-S/II provides interfaces to four languages: APL, BASIC, COBOL, and FORTRAN. A COBOL-like syntax will be used to illustrate the DML's use.

The UWA has eight special status registers. The registers are used by the DBCS and the application program to share information about the status of the database. Here is a list of the registers' names (for COBOL) and an explanation of each:

1. *DB-STATUS*. This register returns the status of the DML statement after its execution. If no error occurs, the DB-STATUS returns zero. For example:
   IF DB-STATUS = 00 Check for a "no error" condition
   (COBOL SENTENCES)
   …
   …
2. *DB-REALM-NAME*. This register returns the name of the realm at the conclusion of DML sentences. Whether the DML's completion is successful or unsuccessful, the realm name is updated. The realm name can be blank. A COBOL program can check the value of this register.
3. *DB-SET-NAME*. This register returns the set name after an unsuccessful DML statement. (It can be blank.) A COBOL program can check the status of this register, but only the DBCS can update it.

FIGURE L.9 INTERACTION BETWEEN THE DBMS AND ROBCOR INFORMATION SYSTEM COMPONENTS



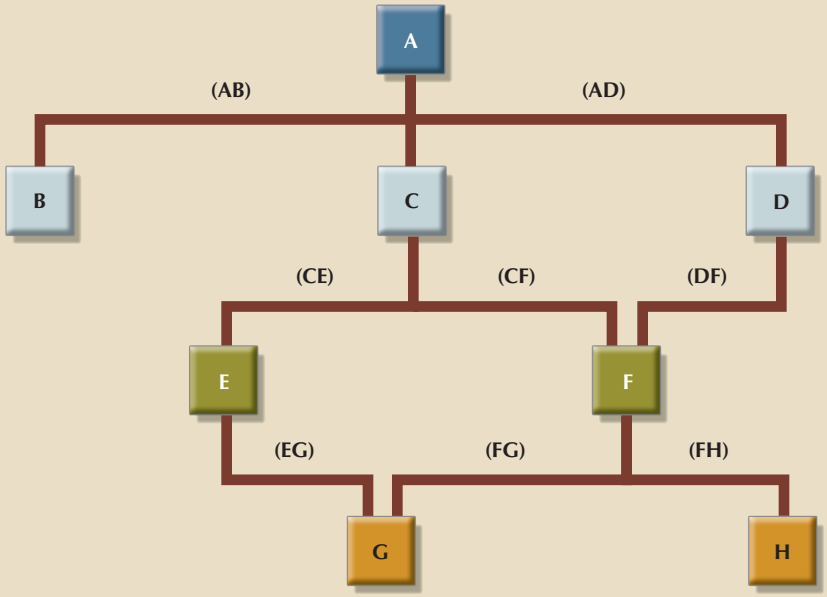
4. *DB-RECORD*. This register returns the record name at the conclusion of DML statements. Whether or not the statement was successful, the DB-RECORD is updated by the DBCS after each statement. DB-RECORD can be set to blank. A COBOL program can check its value.
5. *DB-PRIVACY-KEY*. The DBCS places the value of the PRIVACY key in this register during the schema translation. The PRIVACY key is a keyword used to restrict access to the database's authorized users.
6. *DIRECT-REFERENCE*. This register passes on a record key for DIRECT access. A COBOL command can update this register. The DBCS updates this register with the value of the key for the current record in the last DML statement.
7. *DB-DATA-NAME*. If the subschema was translated with the INCLUDE DATA NAMES option, the DBCS returns the DATA-ITEM name when an invalid data type problem occurs.
8. *DB-KEY-NAME*. The DBCS returns the name of the record key at the conclusion of an unsuccessful DML statement.

To understand the DML statements, it is important that you note the *currency* concept described in Section L-1. So you can appreciate the role played by currency, let's examine an instance of sequential file processing (see Figure L.10) with only one record type.

FIGURE L.10 AN INSTANCE OF SEQUENTIAL FILE PROCESSING WITH ONLY ONE RECORD TYPE

When working with a single record type, there is only one logical path for each record occurrence. If the pointer is located in record A, the next record will always be B, the record after that will always be C, and so on. However, when there are several record types in a network database environment, each record can be involved in more than one relation. That is, a record can be an owner or a member of more than one set. That condition is illustrated in Figure L.11.

FIGURE L.11 SEVERAL RECORD TYPES IN MULTIPLE RELATIONS



In Figure L.11, several logical paths may be taken to navigate from one record to another. For example, if the record pointer is in record A, the path may lead to B (AB set), C (AC set), or D (AD set). If the record pointer is in record F, the path may lead to D (DF set) or C (CF set), or you may elect to move to C (CF set) or H (FH set).

To keep track of the record and set pointers, L-D-S/II keeps five currency register records in the UWA. The currency registers are:

1. *Current record of the run unit*. A pointer updated by the DBCS after certain DML statements. This is the pointer to the last valid record accessed by the application.
2. *Current record of a set type*. A pointer for each set defined in the subschema. Such pointers specify the last record in each set that was accessed by the application.
3. *Current record of a realm*. A pointer for each realm specified in the subschema. Remember that a database can be stored in one or more areas or realms (physical files).

4. *Current record of a record type*. A pointer for each of the subschema's record types.
5. *Current record of a record key type*. A pointer for each record key type defined in the subschema. Each record key type points to a specific record. DBCS keeps a pointer to the last record accessed for each of the defined record key types.

L-8 Data Manipulation Language Commands

The data in a database (especially a transaction-type database) are subject to change. Therefore, end users must be able to add, delete, and modify the data. You will examine how data manipulation is done in a network database environment.

L-8a Opening Realms

An application program must invoke the READY command to access a database. The READY command makes the database available to the program. The three following usage modes are available:

1. UPDATE—Read/write to the database.
2. LOAD—Initial load of the database.
3. RETRIEVAL—Read from the database.

The command syntax conforms to the following sequence:
READY <real name>; USAGE IS {LOAD, UPDATE, RETRIEVAL}

L-8b Closing Realms

When the database's use is no longer required, close the realm using the syntax:

FINISH (realm list)

The realm list refers to the realm names that make up a database. If no realm names are specified, all realms (databases) are closed.

L-8c STORE

The STORE command saves a database record and updates the current record of the run unit in the UWA. The appropriate syntax is:

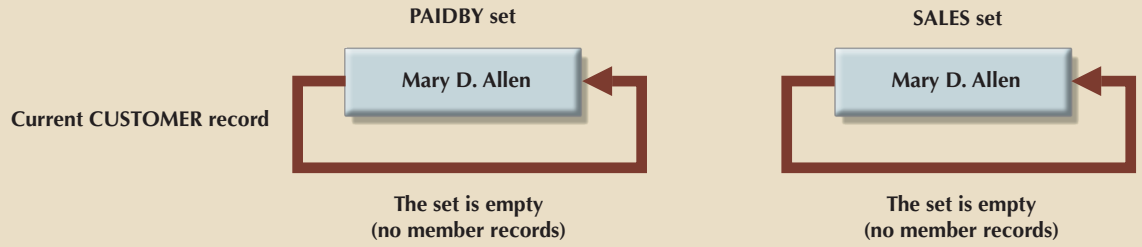
STORE (record-name)

To store a CUSTOMER record in the ROBCOR database, first move the new values to the corresponding fields:

MOVE "12421" TO CUSTID
MOVE "Mary D. Allen" TO CUSTNAME
MOVE "1418 E. Main Street" TO CUSTADDRESS
STORE CUSTOMER

Given that command sequence, the record will become the current record of the run unit, the current record of the PAIDBY and SALES sets, the current record of the realm, and the current record for the CUSTOMER record type, as shown in Figure L.12.

FIGURE L.12 STORING THE CUSTOMER RECORD



The order in which the records are stored is very important. Member records can be stored only after the record owner of the set has been stored. When a member record is stored, it is automatically inserted in all of the sets where the record was declared a member—if *the INSERTION IS AUTOMATIC clause was specified in the schema definition*. For example, a PAYMENT record may be inserted in the ROBCOR database by using:

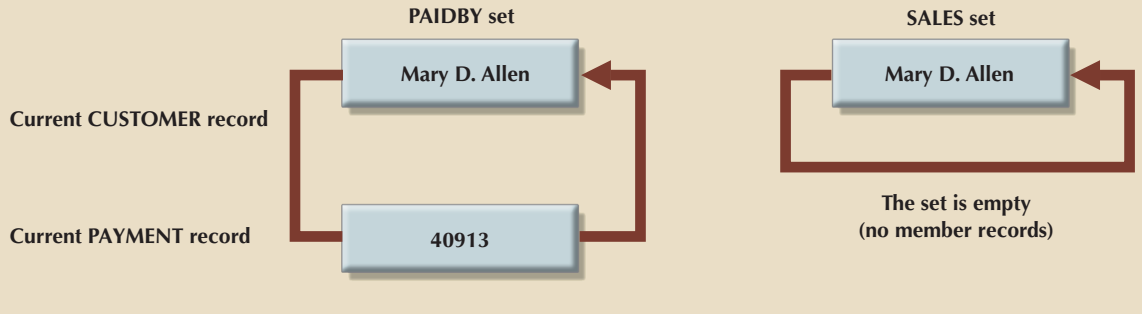
```
MOVE 40913 TO PAYNUM
MOVE "20181029" TO PAYDATE
MOVE 123.00 TO PAYAMOUNT
STORE PAYMENT
```

In that case, the PAYMENT record is inserted into the database; it is also automatically inserted into the PAIDBY set and linked to the CUSTOMER Mary D. Allen CUSTOMER record. The new values of the sets are indicated in Figure L.13.

If a record belongs to more than one set, the programmer must make sure that the current occurrences of the owner records (for the sets to which the new record belongs) are correct before the record is inserted. For example, if the INVOICE record belongs to two sets (SALES and COMMISSIONS), the SALESREP record must be stored prior to the storage of the INVOICE record:

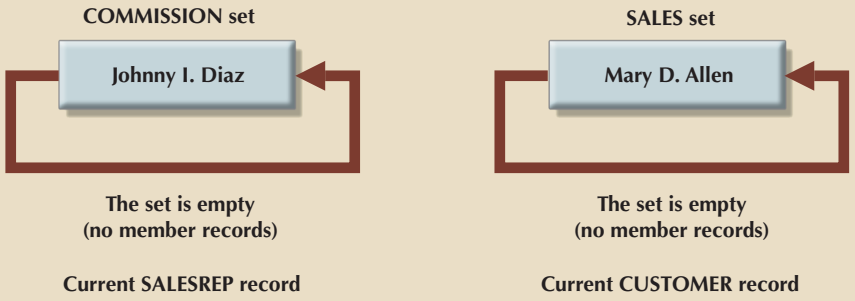
```
MOVE "D234" TO SLSNUM

MOVE "Johnny I. Diaz"
STORE SALESREP
```

FIGURE L.13 THE RESULTS OF THE PAYMENT RECORD INSERTION



After the insertion, the sets are represented as shown in Figure L.14.

FIGURE L.14 THE EMPTY COMMISSION AND SALES SETS

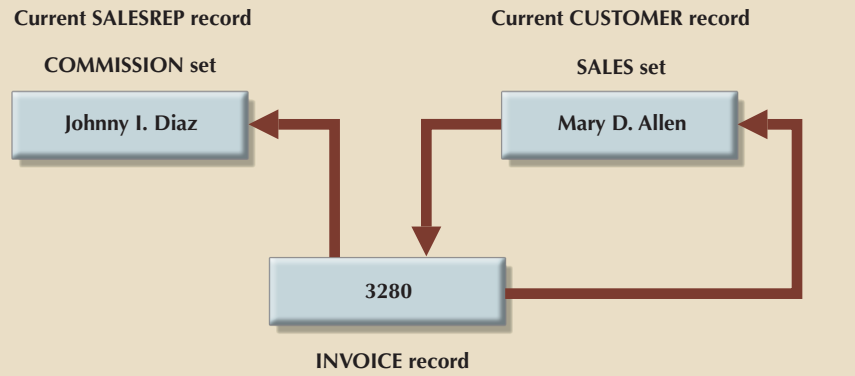


Next, the INVOICE record can be stored:

```
MOVE 3280 TO INVNUM
MOVE "20180114" TO INVDATE
MOVE 169.50 TO INVAMOUNT
STORE INVOICE
```

After that insertion, the sets look like Figure L.15.

FIGURE L.15 THE COMMISSION AND SALES SET AFTER THE INSERTION



L-8d FIND

The FIND command is used to locate records in the database and works with the LOCATION MODE used in the schema definition. The FIND command updates the currency values of the UWA. The syntax for the FIND command varies according to the access type.

Direct Access Mode

The command syntax for the direct access mode is:

```
FIND (record name); DB-KEY IS (dbkey)
```

The dbkey is the DIRECT-REFERENCE special field in the UWA. The direct-reference value must be moved to the DIRECT-REFERENCE field before the record can be accessed. For example:

```
MOVE "101" TO DIRECT-REFERENCE
FIND CUSTOMER; DB-KEY IS DIRECT-REFERENCE
```

The CUSTOMER record type must have been described as LOCATION MODE IS DIRECT in the schema definition.

CALC Access Mode

There are two options. The command syntax for each option follows:

```
FIND ANY <record name>
```

FIND DUPLICATE <record name>

Either FIND locates a record with LOCATION MODE IS CALC. The value to the key field is given before issuing the command:

MOVE "12421" TO CUSID
FIND ANY CUSTOMER RECORD

This command sequence locates the CUSTOMER record "Mary D. Allen."

To use the CALC access mode, the CUSTOMER record must have been defined in the ROBCOR schema as LOCATION MODE CALC USING CUSID. That means that the contents of CUSID are used to find the record.

To store the occurrence of a PAYMENT record in the PAIDBY set, the commands shown in Table L.8 are necessary.

TABLE L.8

STORE THE OCCURRENCE OF A PAYMENT RECORD IN THE PAIDBY SET

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSID	
FIND ANY CUSTOMER	Makes CUSTOMER the current record.
MOVE 40913 TO PAYNUM	
MOVE "20181029" TO PAYDATE	
MOVE 123.00 TO PAYAMOUNT	
STORE PAYMENT	

Navigating Within Sets

There are four options. The command syntax for each option follows:

Find PRIOR <record name> FIRST <set-name>
Find PRIOR <record name> NEXT <set-name>
Find PRIOR <record name> PRIOR <set-name>
Find PRIOR <record name> LAST <set-name>

As the syntax suggests, the FIND command locates the FIRST, NEXT, PRIOR, or LAST occurrence of a given record type within a set. The pseudocode in Table L.9 shows an example.

TABLE L.9

AN EXAMPLE OF THE FIND SYNTAX

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSID	
FIND ANY CUSTOMER RECORD	Locates the owner of a set
IF DB-STATUS NOT = 00	
DISPLAY "CUSTOMER NOT FOUND"	
GO TO ERROR-RTN	
FIND FIRST INVOICE RECORD WITHIN SALES	Locates the first INVOICE in the SALES set for customer 12421
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO ERROR-RTN	

Locating Owner Records

To locate the owner of a member record in a set, use a modification of the FIND syntax:

FIND OWNER WITHIN <set-name>

An example of the command syntax is shown in Table L.10.

TABLE L.10

LOCATING OWNER RECORDS

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSID	Makes CUSTOMER the current record
FIND ANY CUSTOMER	
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO ERROR-RTN	
FIND FIRST INVOICE RECORD WITHIN SALES	
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO "ERROR-RTN"	
FIND OWNER WITHINCOMMISSION	
IF DB-STATUS NOT = 00	Check status
DISPLAY "ERROR"	
GOTO ERROR-RTN	

The command sequence in Table L.10 is a good example of how the application program navigates the database. First, locate the CUSTOMER record 12421 for a customer named "Mary D. Allen." Next, move into the SALES set to locate Mary D. Allen's first INVOICE. Finally, the

FIND OWNER WITHIN COMMISSION

command locates the SALESREP record for that INVOICE.

L-8e CONNECT

The purpose of the CONNECT command is to insert an existing record as a set member. Both the member and the owner records must already be stored in the database. The command syntax is:

CONNECT <record-name> TO <set-name>

The CONNECT command is used when the INSERTION IS MANUAL and the OWNER IDENTIFIED BY APPLICATION clauses were specified for the member record in the schema definition. The user has to manually CONNECT the record with the appropriate owner record in each of the sets to which the record belongs. Assuming the PAYMENT record was defined as INSERTION IS MANUAL in the PAIDBY set, the correct sequence of commands is shown in Table L.11.

TABLE L.11

INSERTING AN EXISTING RECORD AS A SET MEMBER

PSEUDOCODE	COMMENT
MOVE "12421" TO PAYNUM	Makes CUSTOMER the current record
FIND ANY CUSTOMER	
MOVE "40913" TO PAYNUM	
MOVE "20181029" TO PAYDATE	
MOVE 123.00 TO PAYAMOUNT	
STORE PAYMENT	Stores PAYMENT
CONNECT PAYMENT TO PAIDBY	Inserts record in PAIDBY set

L-8f DISCONNECT

The DISCONNECT command removes a record from a set. The command is used only when records were declared as AUTOMATIC OPTIONAL or MANUAL OPTIONAL members of a set in the schema definition. The syntax is:

DISCONNECT <record-name> FROM <set-name>

Table L.12 shows an example.

TABLE L.12

REMOVE A RECORD FROM A SET

PSEUDOCODE	COMMENT
MOVE "40913" TO PAYNUM	Locates the PAYMENT record
FIND ANY PAYMENT	
DISCONNECT PAYMENT FROM PAIDBY	Disconnects the PAYMENT record from the PAIDBY set

Note that the DISCONNECT command in Table L.12 does not physically remove the record from the stored database; it merely manipulates the pointers to bypass the record.

L-8g GET

The GET command reads a record from the database, making the record's field available to the program. Only the fields defined in the subschema are available. The command syntax is:

GET <record-name>

Table L.13 shows an example.

TABLE L.13

READ A RECORD WITH GET

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSTID	Locates the customer record
FIND ANY CUSTOMER	
GET CUSTOMER	Reads the customer's data

L-8h MODIFY

The MODIFY command changes the current record's field contents, using the syntax:

MODIFY <record-name>

The command flushes the contents of the UWA buffers to the database. An example of the MODIFY command is shown in Table L.14.

TABLE L.14

MODIFY A RECORD

PSEUDOCODE	COMMENT
MOVE "12421" TO CUSTID	Locates the customer record
FIND ANY CUSTOMER	
GET CUSTOMER	Reads the customer record
MOVE "245 S.W. Clark St." TO CUSTADDRESS	Changes the address in the UWA buffer
MODIFY CUSTOMER	Writes the changes to the (physical) database

L-8i ERASE

The ERASE command removes the current record from the database *and automatically removes all member records associated with it*. The command syntax is:

ERASE <record-name> ALL MEMBERS

The ERASE command ensures that the record is eliminated from all of the sets in which it was declared a member. If the record was declared owner of one or more sets, all member occurrences related to the record are also removed. Table L.15 shows an example.

TABLE L.15

DELETE A RECORD

PSEUDOCODE	COMMENT
MOVE "14206" TO INVNUM	
FIND ANY INVOICE	Locates the invoice
ERASE INVOICE ALL MEMBERS	Erases the INVOICE record and all member records of all sets from which INVOICE is the owner record

The command sequence shown in Table L.15 will erase the INVOICE records from:

1. All of the sets (COMMISSIONS and SALES) for which it was declared a member.
2. All of the INVLINE records associated with the INVOICE in the INVLINES set.
3. All of the INVLINE members associated with the PRODSOLD set.

L-9 The Network Model's Contribution to Database Systems

The network database model provided several advantages over its file-system and hierarchical-database predecessors. In fact, the network database model paved the way for subsequent database developments through CODASYL's attempt to standardize basic database concepts such as schema, subschema, and DML. The network database model also set the stage for more complex and better data modeling by providing support for relations in which a record could be related to more than one owner or parent record.

Key Terms

AREA, L-9

currency, L-3

Database Administrator Control System (DBACS), L-9

data definition language (DDL), L-5

Interactive Database Processor (IDP), L-5

LOCATION MODE clause, L-9

RECORD NAME clause, L-9

schema, L-5

subschema, L-14

UWA (user work area), L-14