decsystem10

DATA BASE MANAGEMENT SYSTEM PROGRAMMER'S PROCEDURES MANUAL

Order No. AA-0901C-TB

OPERATING SYSTEM AND VERSION: TOPS-10 V6.02, 6.03

SOFTWARE VERSION: DBMS V5
COBOL V11
FORTRAN V5

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation • maynard. massachusetts

First Printing, May 1977

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECSYSTEM-20	TYPESET-11

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

TABLES

PREFACE

This manual describes the DECsystem-10 Data Base Management System (DBMS). The information is addressed to the application programmer who wishes to access the data base.

As an application programmer using DBMS, you should also have a working knowledge of COBOL or FORTRAN, since either may be used as the host language. Descriptions of these languages can be found in the *DECsystem-10 COBOL Programmer's Reference Manual* and the *FORTRAN Reference Manual*, respectively. In addition, you may wish to read portions of the *Data Base Management System (DBMS) Data Base Administrator's Procedures Manual* to understand overall system functioning.

Because DBMS-10 is a CODASYL-based system (a subset of the 1971 Data Base Task Group specification with extensions) you may also wish to read the CODASYL documents to understand data base concepts and implementations.

This manual is intended to complement the Administrator's manual. Chapter 1 deals briefly with concepts and terms and describes a typical DBMS application. Chapter 2 discusses the schema and the keywords associated with it, and illustrates use of the Data Manipulation Language (DML) to access the data base. Chapter 3 defines the conventions used to write DML statements; discusses exception handling; and describes the formats and rules for each DML statement. Chapters 4 and 5 specify the DML use for COBOL and FORTRAN programs, respectively, and provide examples for each.

The appendices include

1. reserved words and user-referencable DBCS terms
2. exception condition codes and error messages
3. schema data declarations and FORTRAN and COBOL conversions
4. passing strings to DBCS
5. a glossary.

The glossary briefly defines DBMS-unique terms.

CHAPTER 1
INTRODUCTION TO DBMS

The Data Base Management System (DBMS) is a framework for creating, maintaining, and referencing groups of interrelated data. These groups – called data bases – have been structured and linked in a way that permits you to selectively access and manipulate the data in the data bases. Responsibility for defining, organizing, protecting, and documenting the data base itself lies with the Data Base Administrator (DBA).

The main elements of data base management software comprise:

- DDL the data description language and its processor. Refer to Section 1.3 and to the *DBMS-10 Data Base Administrator's Procedures Manual.*
- DML the data manipulation language for COBOL and FORTRAN programs. The main portion of this manual is devoted to explaining the DML and its use.
- DBCS the data base manager module of reentrant run-time routines. Refer to Section 1.5 and to the *DBMS-10 Data Base Administrator's Procedures Manual.*
- DBU the data base support utilities. Refer to the *DBMS-10 Data Base Administrator's Procedures Manual,* Chapter 6.

1.1 DBMS FEATURES

DBMS allows the building and manipulation of data structures that are complex or simple, depending on the needs of your application. Some of the advantageous features of this system are reduced program-development time, data integration, and simplified program maintenance.

1.1.1 Reduced Program-Development Time

DBMS allows you to call generalized run-time routines to handle many program-development functions in a standard way thereby reducing the amount of time required to design and code an application.

Many application programs require data structures more complex than a simple sequential series of records. They may require randomized data, direct access to data, and a method of allowing one record to point to another. Many application projects therefore recode the same basic routines for access methods and pointer handling. Because these requirements are common to many application projects, however, standardization has been possible with DBMS.

1.1.2 Integration of Data

Since DBMS can support tree and network data structures and multiple-access methods, one physical data base can serve many applications. This integration minimizes data redundancy and its resultant multiple-update problems.

1.1.3 Simplified Program Maintenance

With DBMS, an application program is isolated from the structure of data formats by the schema. (The run-time system (DBCS) performs a binding operation which, in effect, causes the required formats and data description information to be copied into the source program.) Functionally, this centralizes required data format changes to one location and renders them purely mechanical.

As an application programmer, you have traditionally been required to develop and specify the data descriptions, record formats, and buffer sizes for your programs. When changes occur in these descriptions, formats, or buffer sizes, you edit and recompile the sources. Moreover, applications that require data structuring (creation of pointers that relate records to one another) or special search techniques have the code for implementing these functions embedded in the program. Changes in record sizes or formats for these applications can invalidate the structure or search techniques and require that the program be rewritten. With DBMS many of these maintenance functions become simple and mechanical.

The next section describes terms associated with data-base management to familiarize you with their use in this manual.

1.2 DBMS-ASSOCIATED TERMS

The description of a data base is made using the names and characteristics of such terms as data-items, data aggregates, records, areas, and sets. Because some of these terms are used in a special way in DBMS, their definitions and implications are important to you. As such, they are described in some detail here. Those terms describing aspects of DBMS function that primarily concern the Data Base Administrator are not emphasized – although they are discussed.

A DATA-ITEM is the smallest unit of named data. An occurrence of a data-item is a value. Data-items can be alphanumeric or numeric (fixed or floating point).

A DATA-AGGREGATE is a named collection of data-items within a record. Aggregates are of two types: vectors and repeating groups. A vector is a one-dimensional, ordered collection of data-items, all of which have identical data types. A repeating group is a collection of data that occurs an arbitrary number of times within a record occurrence. The collection may consist of data-items, vectors, and repeating groups.

A RECORD is a named collection of one or more data-items or data-aggregates. A data base can contain any number of occurrences of each record described in the schema. Each record described defines a record type. (For example, for each employee in a company using DBMS, the data base would contain one occurrence of the record type PAYROLL-RECORD.) This distinction between the actual occurrences of a record and the type of the record is an important one for DBMS users.

A SET is a named collection of record types. As such, it establishes the characteristics of an arbitrary number of occurrences of the named set. Each set type specified in the schema must have one record type declared as its OWNER and one or more record types declared as its MEMBER records. Each occurrence of a set must contain one occurrence of its owner record and may contain an arbitrary number of occurrences of each of its member record types.

An AREA is a named sub-division of the addressable storage space in the data base and can contain occurrences of records and sets. Areas can be opened by a run-unit with USAGE MODES which permit or forbid concurrent run-units to open the same area. An area can be declared in the schema to be a TEMPORARY AREA. This provides a different occurrence of the temporary area to each run-unit opening it. At the termination of the run-unit, the storage space involved becomes available for re-use; any data stored in the temporary area is lost.

Use of the area concept allows the Data Base Administrator to divide a data base and to control placement of an area for efficient storage and retrieval of data. It allows efficient access to the data base (since each run-unit uses only a specified portion of the data base); and a convenient unit for recovery (since duplication and backup can be carried out selectively).

A SCHEMA, defined by the Data Base Administrator, consists of Data Description Language entries and is a complete description of a data base. It includes the names and descriptions of all the areas, sets, records, and associated data-items and data-aggregates that compose the data base.

A SUB-SCHEMA, also defined by the Data Base Administrator, consists of DDL entries delineating those areas, sets, records, and associated data-items and data-aggregates known to an application. Descriptions are in the form known to the application using the sub-schema.

A DATA BASE consists of all the record occurrences, set occurrences, and areas defined by its schema. Each data base has its own schema. The contents of different data bases in a facility are disjoint.

1.3 DBMS LANGUAGES

The DBMS provides four languages – each having a different function:

- Schema Data Description Language (DDL)
- Sub-Schema Data Description Language (DDL)
- Device Media Control Language (DMCL)
- Data Manipulation Language (DML)

The Data Base Administrator uses the DDLs and the DMCL; therefore they are not treated in detail here. You use the DML – in conjunction with a host language – to access the data base. The body of this manual is therefore devoted to describing the DML and illustrating its use.

1.3.1 Data Description Languages (DDLs)

The Data Base Administrator uses the DDLs to define the schema and the sub-schema. The Schema DDL enables the DBA to describe the overall logical and physical mapping of the data base. The Sub-Schema DDL allows him to describe a specific subset of the data base to be accessed by an application.

1.3.2 Device Media Control Language (DMCL)

The DBA uses the DMCL to specify how the physical storage space on mass storage devices will be used to record the data base.

1.3.3 Data Manipulation Language (DML)

As the application programmer, you use the DML to access data in the data base. The DML is not a complete language. Rather, it is a host-language extension relying on COBOL or FORTRAN to offer a framework; from this the DML can provide the interface with the data base. The DML commands and the host-language statements coexist in an application program. The distinction between them is merely conceptual. From your point of view, you are using one unified language that has the capabilities of both the host language and the DML. The host language manipulates data in primary storage, and the DML interfaces with the data base. All calls to retrieve data, to add new data, to modify existing data or data relationships, and to delete existing data or data relationships in the data base are written in the DML.

The main body of this manual describes the DML, its use, the commands associated with it, and its relation to COBOL and FORTRAN. Chapter 3 defines the specifications for the DML.

1.4 UNDERSTANDING DBMS RECORDS AND SETS

A record description in the Schema DDL is similar to a COBOL record description. It is a named collection of data-items or data-aggregates. The DBA describes the record when he creates the data base. A record from the data base looks exactly like a COBOL data record to the application program and can be treated as such. The difference is that you do not describe the records in your program. Rather the compiler or preprocessor inserts these records into your program.

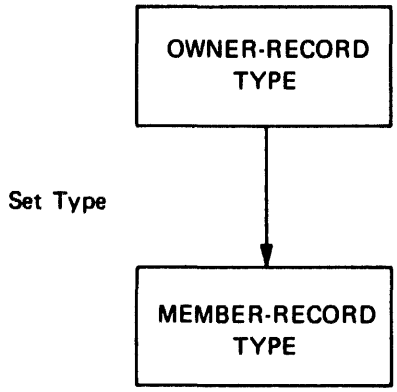
Records are organized into sets within the data base. A set is a named collection of records. Sets have one owner record and can have one or more member records.

A set is normally arranged for a specific application; if other applications use some of the data in that set, another set can be created to include that data in an arrangement suitable for these applications. This avoids redundancy of data and provides ease of accessing the data required for a given processing task.

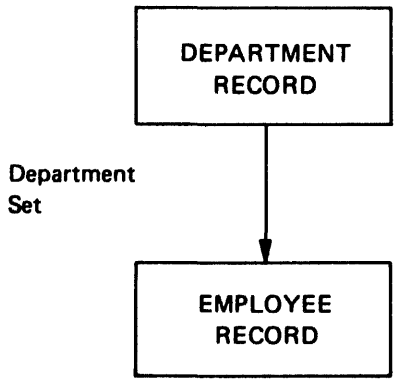
For example, suppose a company has a data record for each employee. One set of records used by the Personnel Department contains the names of the employees in each department. The Personnel Department or the department manager is the owner in this set and each employee in the department is a member. The Payroll Department needs a different kind of set, however, because its applications are different. Its set may consist of wage class as the owner record and the employees in that wage class as members. The Accounting Department can then use the same employee records grouping them into a different type of set. This set consists of an employee record as the owner and his dependents as members. If an employee has no dependents, the set has an owner record but no member records. All company applications that use employee records can use the same data records. The records are logically arranged into different sets, however, to meet the needs of each application.

1.4.1 Types of Sets and Records

The description of a set in the schema defines a set type. The description of a record in the schema defines a record type. The description of a set type is a named collection of record types. A set type consists of an owner-record type and one or more member-record types. Schematically, a set type can be represented as:



A department set type can therefore be represented as:



1.4.2 Occurrences of Sets and Records

A collection of one or more logically related record occurrences defines a set occurrence; this is the actual data in the set. You are primarily concerned with occurrences of sets and records - since your interest is processing the actual data and not the structure of the data base. A set occurrence consists of an owner-record occurrence and zero, one, or more member-record occurrences linked in some way. Schematically, then, a set occurrence can be represented as it is in Figure 1-1.

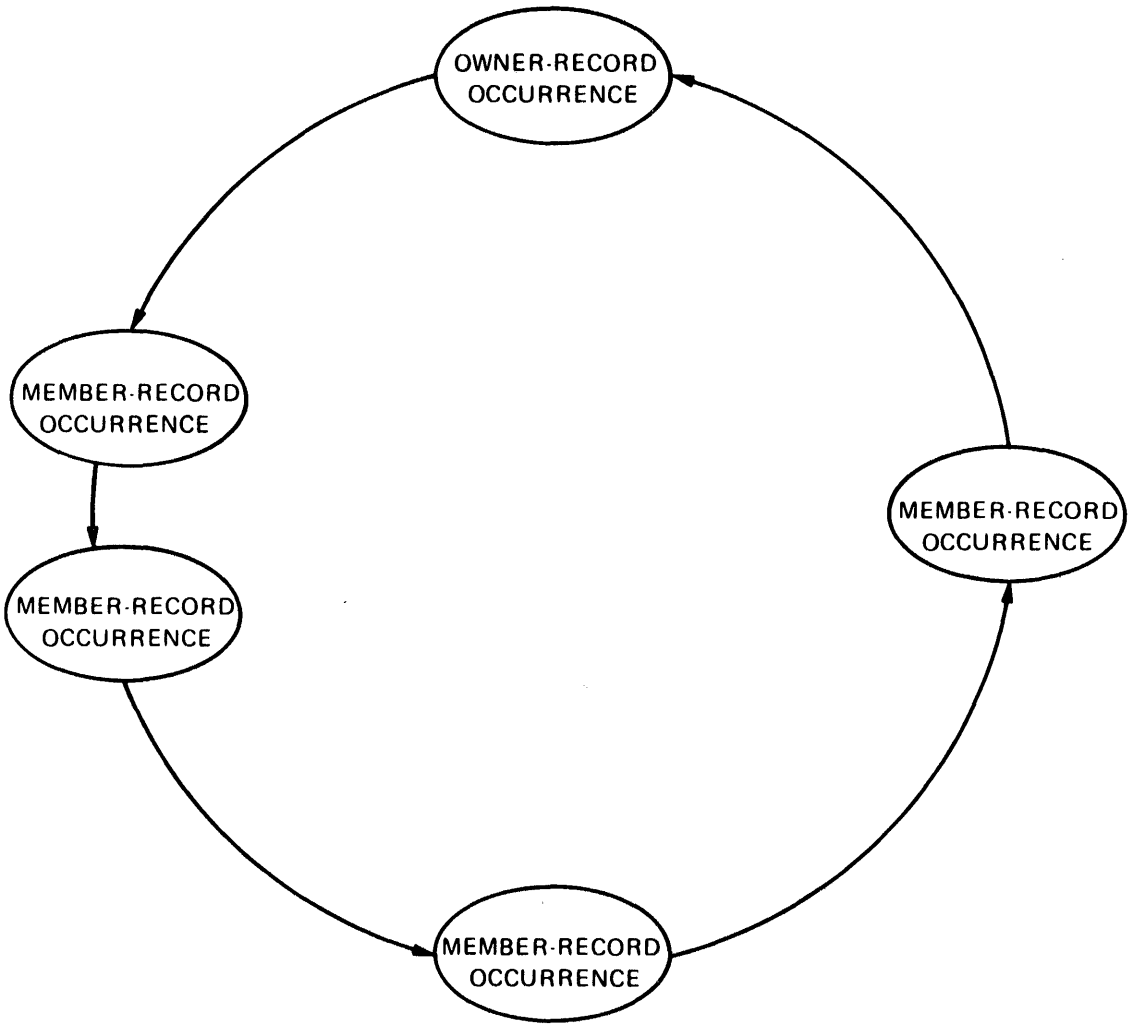


Figure 1-1 Generalized Representation of Set Occurrence

A record occurrence is a specific record of a record type. For example, John Smith's employee record is a specific record occurrence of the employee-record type. The Drafting Department set containing the names of the employees in that department constitutes a set occurrence. The Drafting Department set occurrence can be illustrated schematically as done in Figure 1-2. It looks this way:

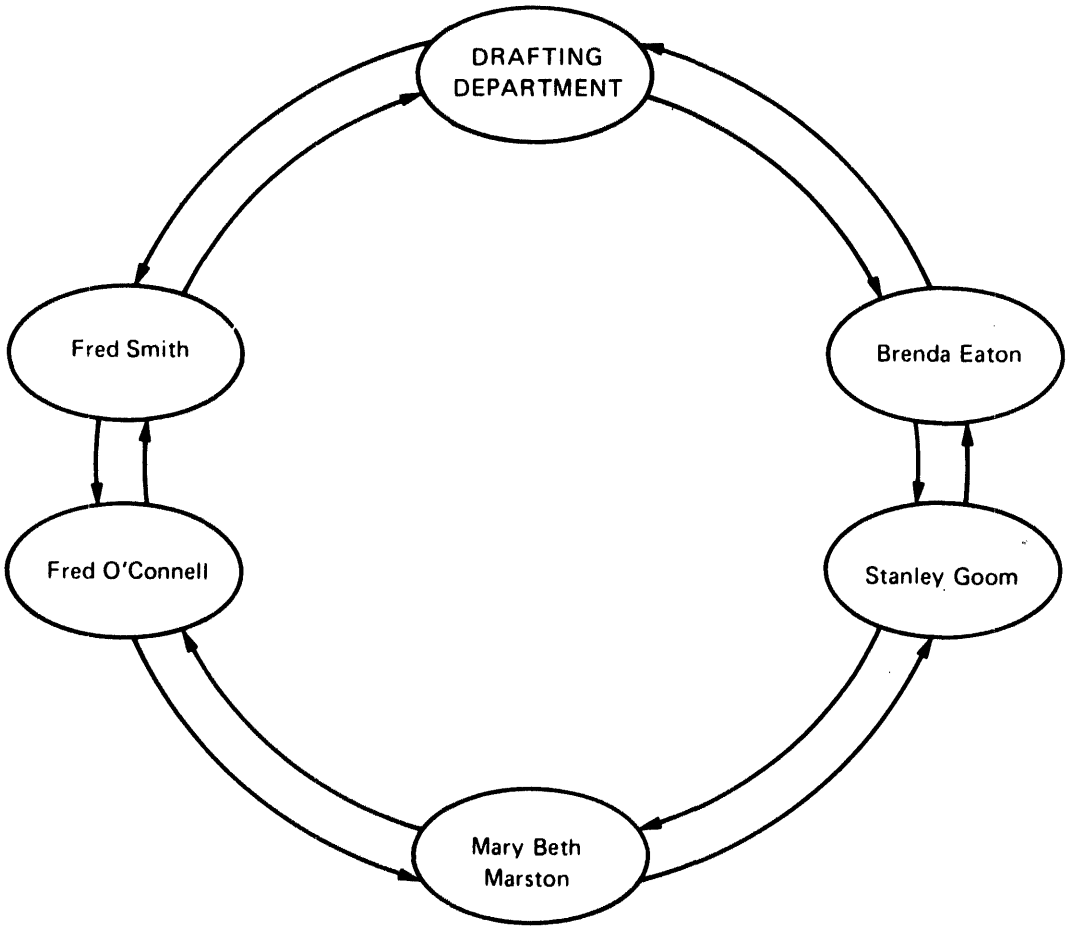


Figure 1-2 Specific Set Occurrence: the Drafting Department

The arrows in the figures show the way the records in the set occurrence are linked. For each set occurrence a chain of pointers is created that provides for serial access to all records in the set occurrence. These pointers are embedded in the records themselves; they link one record in the chain to the next record in the chain. From any given record in the chain, processing can be forward, backward, or directly to the owner record – depending on the linkage definition the DBA indicates in the schema.

In the set occurrence illustrated by Figure 1-2, the arrows indicate that the records are linked in the forward (NEXT) direction and in the backward (PRIOR) direction. Records could also be linked to the owner (LINKED TO OWNER); this would be shown as an arrow from a member record to the owner record.

1.4.3 Main Characteristics of Sets and Records

The following items further characterize sets and records.

1. No intrinsic limitation exists on the number of set types that can be declared in a schema.
2. Each set type must be defined by an owner-record type. It can have one or more member-record types.
3. A record type cannot participate as both owner and member of the same set type.
4. A record type can be declared as the owner record of any number of set types. Likewise, a record type can participate as a member record in one or more set types. Furthermore, a record type can be the owner of one or more set types and – at the same time – a member in any number of set types.
5. A record occurrence cannot appear in more than one occurrence of the same set.
6. A set occurrence is a collection of one or more logically related record occurrences. Each occurrence of a set includes an owner record occurrence and zero, one, or more member-record occurrences.
7. A singular set is one in which the owner is the system.

1.4.4 Set Relationships

Three data structures are used in DBMS to show set relationships: sequential, tree, and network. Refer to Figure 1-3 a, b, and c, respectively, for a schematic representation of these three data structures.

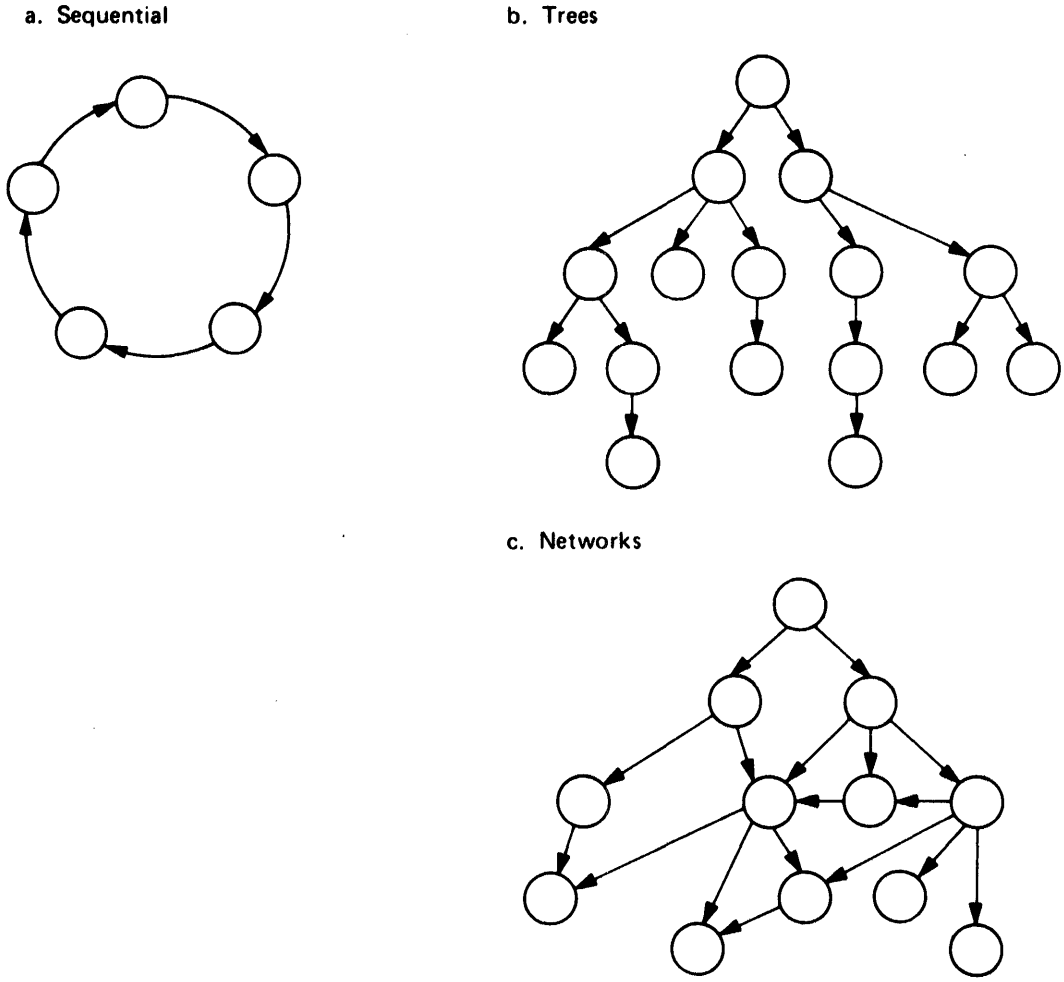


Figure 1-3 DBMS Data Structures: Sequential, Tree, and Network

1.4.4.1 Sequential Structures – A sequential structure shows intra-set relationships. Each element (record) in a DBMS sequential structure is related to the element following it in the structure. The simplest representation of a DBMS sequential structure is a set occurrence linked in the forward direction (with NEXT pointers, Figure 1-3a). This, in effect, is a one-way ring. If linkages in the backward direction are included (that is, PRIOR pointers), the set occurrence is a two-way ring. (See Figures 1-1 and 1-2 for a detailed representation of set occurrences – a basic construct in DBMS). Linkages to the owner record from each member record provide a further facility for processing.

1.4.4.2 Tree Structures – A tree structure shows inter-set relationships. A tree data structure is hierarchical; each element is related to one or more elements at any level below it, but only to one element in the level immediately above it. The highest element of the tree is called the root; and it has only dependent elements. Each node of the tree, then, has one branch entering it, but may have any number (including zero) exiting. (Figure 1-5 shows sets in a tree structure describing the application discussed in Section 1.6.)

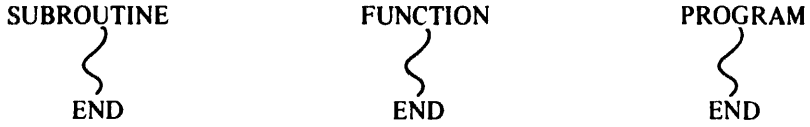
1.4.4.3 Network Structures – A network structure also shows inter-set relationships. The network is the most general form of data structure. In such a structure, any given element may be related to any other element in the structure. Unlike a tree, no restriction exists on the number of branches entering a node. Networks are the most widely used data base structures available in DBMS.

1.5 OPERATIONAL ENVIRONMENT

This section describes those aspects of the DBMS operational environment that are of particular interest to you as the application programmer. These include the run-unit, the Data Base Control System (DBCS), the User Working Area (UWA), and the techniques used to ensure protection of data in shared data bases.

1.5.1 Run-Unit

A run-unit is the execution of a program. A program consists of one or more program-units. Program-units are the smallest collection of source-language statements that can be independently compiled or preprocessed. For example, a FORTRAN program-unit is defined as one of the following:



1.5.2 Data Base Control System (DBCS)

The Data Base Control System (DBCS) is the set of routines called by the run-unit's DML statements to perform the data manipulation functions. Figure 1-4 illustrates the general relationships between the data-base storage areas, the DECsystem-10 monitor, a program (run-unit), and its use of the object-time system and DBCS object-time modules.

1.5.3 User Working Area (UWA)

The User Working Area (UWA) is the storage space allocated to a program-unit and serves as the communication medium between the DBCS and the program-unit. The UWA can be considered a loading and unloading zone in which all data provided by the DBCS in response to a call for data is delivered, and where all data to be picked up by the DBCS must be placed. Each program-unit is assigned its own UWA; the data in the UWA is not disturbed except in response to the execution of a DML command or a host-language command. The DBCS creates the UWA for a specific program-unit according to the sub-schema invoked by that program-unit. Only those data-items known to the sub-schema can exist in the UWA, and only those can be referenced by the program-unit. For further discussion of the UWA, refer to Section 2.2.1.

1.5.4 Protection of Data

The DBMS includes provisions for protecting data in data bases shared by many programs and applications. Essentially the system offers two kinds of protection:

1. Privacy, which is protection against unauthorized access to data, and
2. Integrity, which is safeguarding of data from destructive interaction of program run-units.

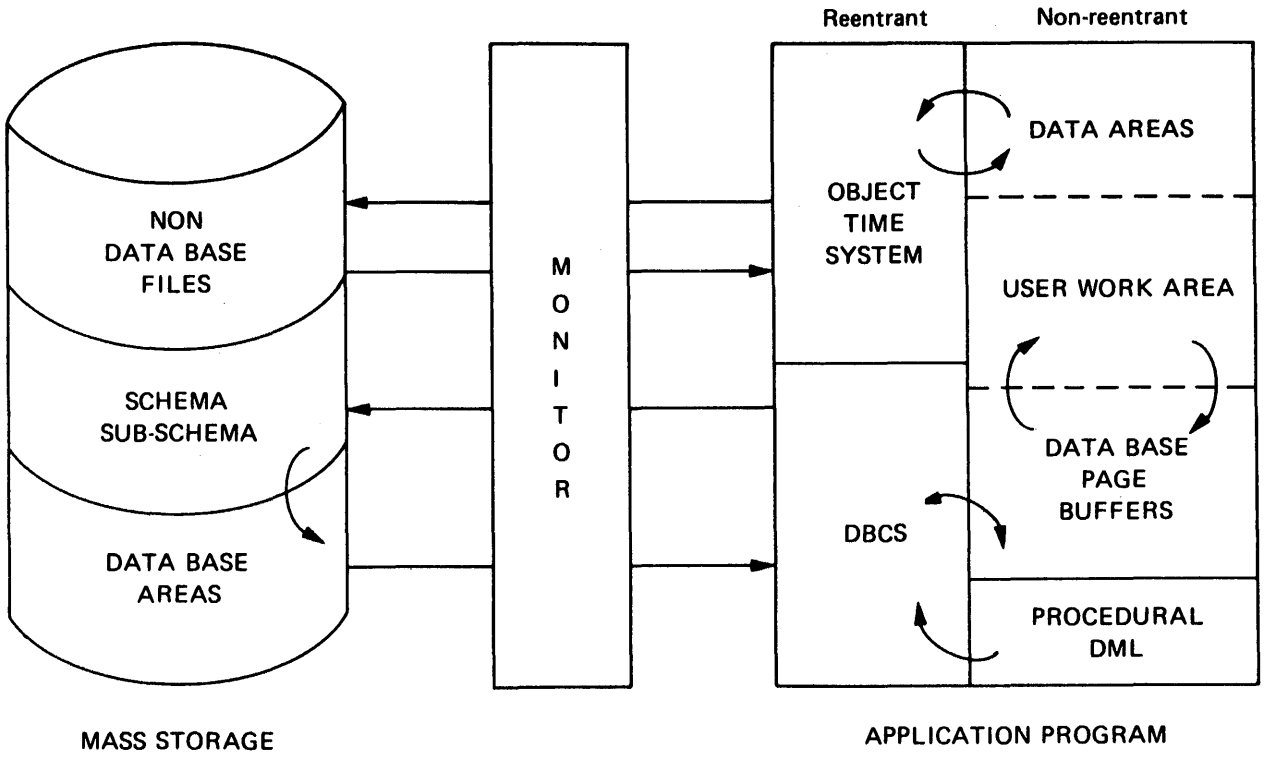


Figure 1-4 Data-Base Environment

Note, however, DBMS offers no protection against your deleting a record that is also owner of a set having members required by other programs. This type of deletion is considered a logical error and must be guarded against by good administrator–programmer communication in individual facilities.

1.5.4.1 Privacy of Data – Protection against unauthorized access of data in a shared data base is enhanced by the mechanism of privacy locks specified in the schema and sub-schema and privacy keys that must be provided by a run-unit seeking to access or alter the data. A privacy lock is a single alphanumeric value up to five characters long; it may be declared at the sub-schema and area levels. A privacy key is a value supplied by the run-unit seeking access. The system compares the values. If there is no match, the system forbids the run-unit specifying the key access to the data. It returns an exception code to the run-unit.[1] (Refer to Section 3.2 for a discussion of exception handling and a list of statement codes, and to Appendix B for a description of the exception codes.)

To access data, therefore, you must know the value of the key permitting entry into the part of the data base you need. You should also remain current since the DBA may periodically modify or change values for privacy locks.

1.5.4.2 Integrity of Data – Protection of data against concurrent destructive interaction by two run-units is ensured by giving a run-unit exclusive update rights over one or more areas. A concurrent run-unit cannot then access these areas for update.

You must specify the usage mode describing how the data is to be accessed when you open an area. DBMS provides six usage modes: RETRIEVAL, UPDATE, PROTECTED RETRIEVAL, PROTECTED UPDATE, EXCLUSIVE RETRIEVAL, and EXCLUSIVE UPDATE. Refer to Section 2.2.3 for more detailed information on opening and using areas and to Chapter 3 for a specification of the OPEN statement.

[1] The term exception refers to error status conditions. The term is used in conformance with CODASYL terminology.

1.6 A TYPICAL DBMS APPLICATION

The following example of a typical DBMS application is given to further familiarize you with the basic elements of the system. The example analyzes a company's activities and builds a data base using DBMS. Refer to Figure 1-5 as you read the following explanation. The figure illustrates the application showing the sets in a tree structure.

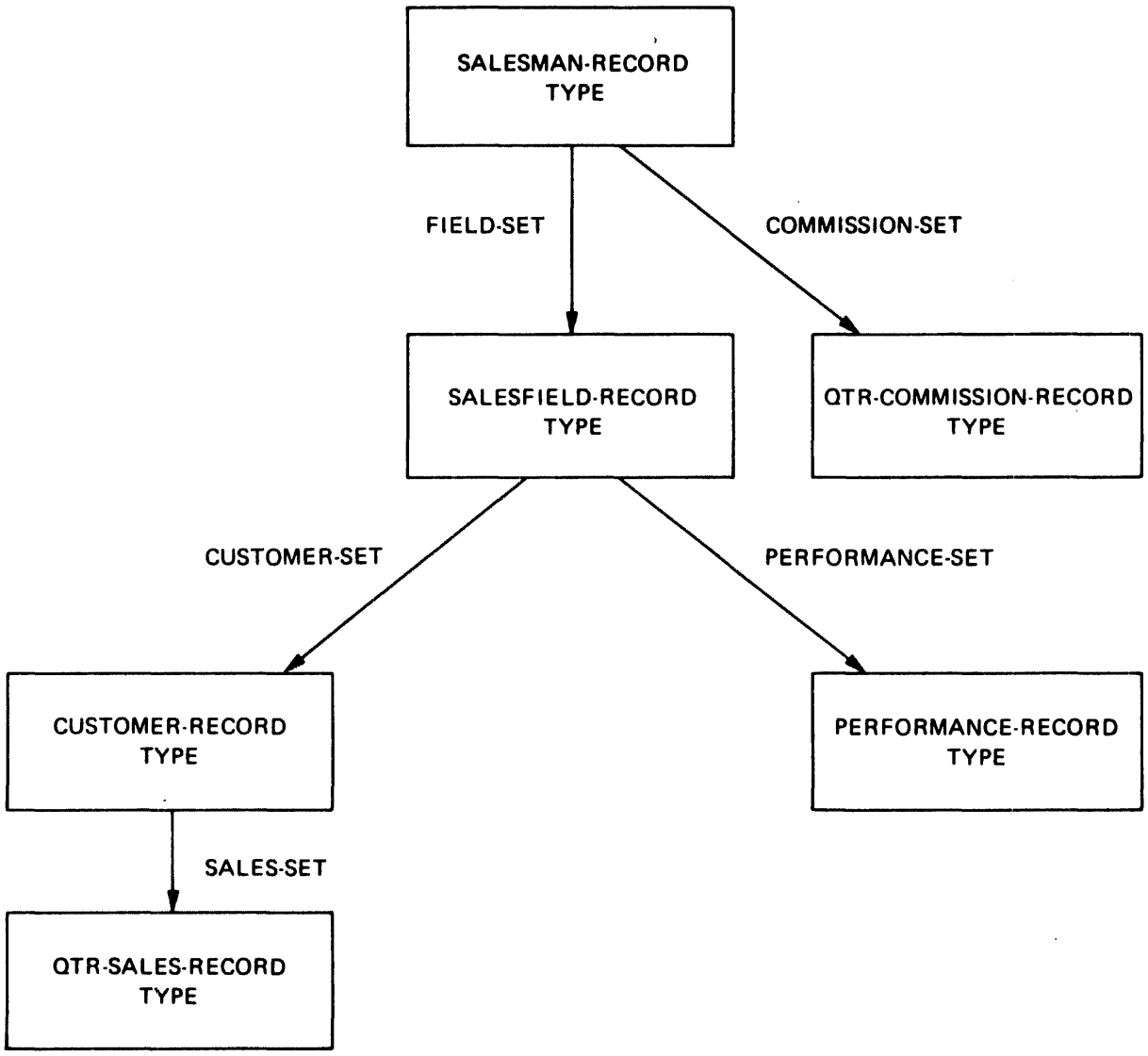


Figure 1-5 BARH Ltd. Application Showing Sets in a Tree Structure

Suppose the existence of a company, a sales organization named BARH Ltd. The company has customers and salesmen. A group of customers serviced by a salesman is a sales territory, and a number of sales territories make up a sales office. One record type named SALESMAN-RECORD describes information relevant to all the salesmen in the company. The data-items in this record include such information as salesman's name, address, phone number, Social Security number, number of dependents, base salary, and hiring date. The record type SALESMAN-RECORD has as many record occurrences as there are salesmen in BARH, Ltd.

Similarly, another record type named CUSTOMER-RECORD contains data-items pertaining to all the customers of the company. Specific record occurrences of this record type include such data as a specific customer's name, address, phone number, account number, and credit status. A third record type SALESFIELD-RECORD has occurrences consisting of such data as a specific territory identification and location of the territory.

The records described are then grouped into set types relevant to the BARH Ltd. application. One set type is FIELD-SET, which consists of the SALESMAN-RECORD as the owner and SALESFIELD-RECORD as an optional member. Using optional membership allows you to conditionally link a salesman with a sales territory and to conditionally change the linking between the salesmen and sales territories.

A further analysis of the company shows that a relation also exists between the customers and the sales territory; this is represented by another set type called CUSTOMER-SET, in which the owner record is SALESFIELD-RECORD; the optional member is CUSTOMER-RECORD. Using optional membership here allows you to link the individual customers with the appropriate sales territory to which they belong.

Like most companies BARH makes predictions about how much income will be earned in each sales territory each quarter; it then compares the actual performance at the end of the quarter with the predicted performance. To define this function, another record type PERFORMANCE-RECORD is necessary. It has an occurrence for each sales territory. Additionally, another set type is required that relates SALESFIELD-RECORD as the owner to the new record as a mandatory member. This set is called PERFORMANCE-SET. Each occurrence of PERFORMANCE-SET then has one occurrence of its owner record and one occurrence of the member record. Note that SALESFIELD-RECORD participates in two other sets – once as owner and once as optional member – but can still participate as the owner in this set.

For each quarter BARH also measures the amount of sales made to each customer and the amount of commission earned by each salesman. Two records define this function: QTR-SALES-RECORD and QTR-COMMISSION-RECORD. Each of these records is used in a new set type. One set is SALES-SET; it has CUSTOMER-RECORD as its owner and QTR-SALES-RECORD as a mandatory member. The other set is COMMISSION-SET and has SALESMAN-RECORD as its owner and QTR-COMMISSION-RECORD as a mandatory member.

Figure 4-2 illustrates the schema and sub-schema for the BARH Ltd. application using COBOL as the host language; Figure 5-2 illustrates the schema and sub-schema using FORTRAN.

CHAPTER 2
USING THE DATA BASE

To write programs that access a data base, first obtain a copy of the data base schema and the sub-schema of that portion of the data base you want to access. By reading the schema and sub-schema, you can find the names of the areas, sets, records and data-items to reference in your program. Section 2.1 contains the detailed information necessary to understanding the schema and sub-schema listing shown in Figure 2-1.

To access the data base, you must include Data Manipulation Language (DML) statements in your program. Section 2.2 introduces the DML statements and illustrates their use. Chapter 3 discusses the DML formats and rules.

To create a journal for backup and recovery, you should know the types of journaling available and the various subroutines associated with journaling. Section 2.3 discusses journaling.

To understand overall system function, you should also know the control DBCS maintains during application-program execution. Section 2.4 discusses conditions under which a program can be returned to monitor level; it also discusses conditions under which the data base is considered to be in an undefined state.

Finally, to use the DBMS efficiently, you should also know which functions the Data Base Control System (DBCS) performs for you; that is, automatically. These functions can affect the number of operations performed on the data and the running time of your program. Section 2.5 discusses these functions in terms of efficient use.

2.1 READING THE SCHEMA AND SUB-SCHEMA

To see the descriptions of the areas, sets, records, and data-items you will access in your application program, examine a listing of the schema and the sub-schema. Refer to Figure 2-1 for an example of a schema, named V4S, with three sub-schemas: SS1, SS2, SS3.

Note that the schema provides a description of each record type and includes the data-items each record type contains; it also provides the key-fields you can use in accessing these records.

Since your application program will gain access to the data base through a sub-schema - using the DML INVOKE statement to call the particular sub-schema you want -- you must carefully study the sub-schema section. The sub-schema lists the areas, records, and sets your program can access. It also includes the privacy lock for which your program must supply a privacy key to gain access to the data base at compile time.

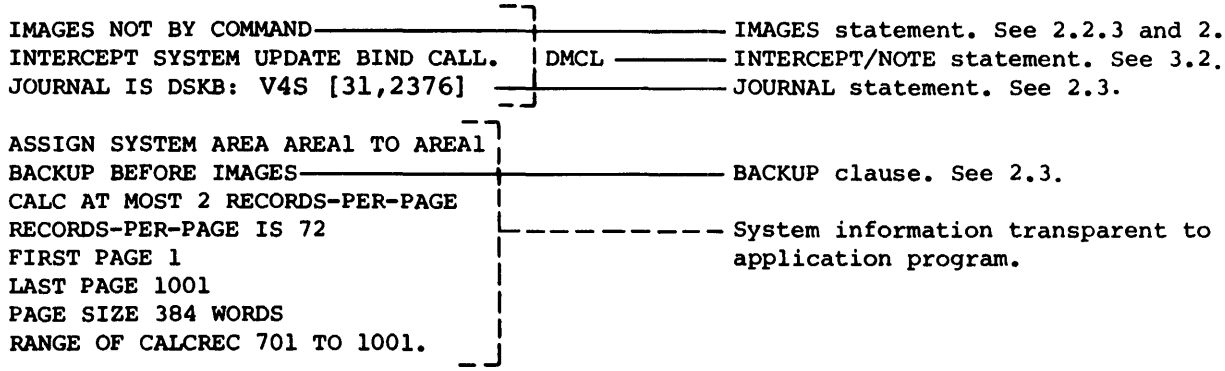


Figure 2-1 Sample Schema and Sub-Schema Listing

```
   SCHEMA NAME IS V4S.                                  Schema name, See 2.2.1.

                                                        Area name used in OPEN and CLOSE.
   AREA NAME IS AREA1 PRIVACY LOCK ABC.                 See 2.2.3 and 2.2.7. Privacy lock.
                                                        See 1.5.4.1 and 2.2.3.

   RECORD NAME IS CALCREC                               Record name.
   LOCATION MODE IS CALC USING PROFES,                  Location mode for CALCREC. Used
   LNAME DUPLICATES NOT ALLOWED WITHIN AREA1.           in storing the record. See 2.1.1.
   02 VACATION%VACAMI TYPE FLOAT BIN.
   02 UNAME PICTURE X(12) USAGE DISPLAY-7.               Data items in CALCREC.
   02 FAMILY SIZE IS 3 WORDS.
   02 PROFES TYPE FIXED BIN.

   RECORD NAME IS SORTREC                               Record name.
   LOCATION MODE IS VIA SYS-SET                         Location mode for SORTREC used in
   WITHIN AREA1.                                        storing the record. See 2.1.1.
   02 EXPER TYPE FIXED DEC 3.
   02 SKILLMASK%SKMASK TYPE FIXED BIN 70.                Data items in SORTREC.

   SET NAME IS SYS-SET                                  Set name.
   ORDER IS SORTED DUPLICATES ARE ALLOWED               Set order. See 2.1.3.
   OWNER IS SYSTEM                                      Owner for SYS-SET. See 2.1.5.
   MODE IS CHAIN.                                       Set mode for SYS-SET. See 2.1.2.
   MEMBER IS SORTREC MAND AUTO                          Member records in SYS-SET and their
   ASC KEY IS SKILLMASK ASC RANGE KEY IS EXPER.         set membership. See 2.1.4.

   SET NAME IS CALCSORT                                 Set name.
   OWNER IS CALCREC                                     Owner for CALCSORT.
   ORDER IS ALWAYS NEXT                                 Set order. See 2.1.3.
   MODE IS CHAIN.                                       Set mode for CALCSORT. See 2.1.2.
   MEMBER IS SORTREC OPTIONAL AUTO LINKED TO OWNER      Set membership.
   SET SELECTION IS LOCATION MODE OF OWNER.

   SUB-SCHEMA NAME IS SS1.                              Sub-schema name used in INVOKE.
                                                        See 2.2.1.

AREA SECTION.
   COPY AREA1.                                          Area from schema included in this
                                                        sub-schema.

RECORD SECTION.                                         Records from the schema included in
   01 SORTREC.                                          this sub-schema.
   01 CALCREC.
   02 FAMILY.
   DATA FAMILY/10,20,30/                                Exact text that will be included in
                                                        host-language program.
   COPY OTHERS.                                         Also copies all other 02 data (i.e.,
                                                        02 VACATION; 02 LNAME; 02 PROFES).
                                                        See CALCREC description.

SET SECTION.                                            Sets from the schema included in
   COPY ALL SETS.                                       sub-schema SS1.
   SUB-SCHEMA NAME IS SS2.                              Sub-schema name used in INVOKE.
                                                        See 2.2.1.

AREA SECTION.                                           Areas from schema included in sub-
   COPY AREA1.                                          schema SS2.
```

Figure 2-1 (Cont.) Sample Schema and Sub-Schema Listing

```
RECORD SECTION.                                  Records from schema included in sub-
   01 SORTREC.                                   schema SS2.
   01 CALCREC.
   02 FAMILY.
   03 A PIC 9(10) COMP VALUE 10.                 Exact text that will be included
   03 B PIC 9(20) COMP VALUE 10.                 in host-language program.
   03 C PIC 9(30) COMP VALUE 10.
   COPY OTHERS.

SET SECTION.
   COPY ALL SETS.

   SUB-SCHEMA NAME IS SS3.                       Sub-schema name used in INVOKE.
                                                 See 2.2.1.

AREA SECTION.
   COPY    TEMPORARY AREA1.                      Area from schema included in sub-
                                                 schema SS3.

RECORD SECTION.                                  Records from schema included in sub-
   01 SORTREC.                                   schema SS3.
   01 CALCREC.
   02 FAMILY.                                    Exact text that will be included
   DATA FAMILY/10,20,30/                         in the host-language program.
   02 LNAME.
   DATA LNAME/'MISTERCALC'/
   COPY OTHERS.                                  See RECORD SECTION SS1.

SET SECTION.
   COPY ALL SETS.                                Sets from schema included in sub-
                                                 schema SS3.
   END-SCHEMA.
```

Figure 2-1 (Cont.) Sample Schema and Sub-Schema Listing

Figure 2-2 represents the set types for the sample schema and sub-schema shown in Figure 2-1. This kind of set representation can be very helpful to you particularly as you begin to "walk through structured data" (as described in Section 2.2.4). You should, therefore, either draw a set representation or request one from the Data Base Administrator.

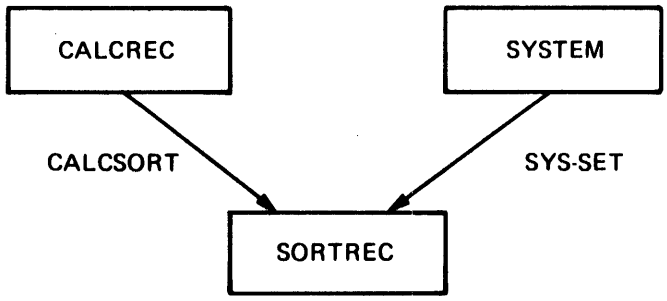


Figure 2-2 Set Representation for Sample Schema

2.1.1 Location Mode

As shown in the sample schema, each record has a designated location mode. The location mode is defined by the Data Base Administrator using the Schema DDL and specifies to the DBCS the criteria for storing and accessing the record. You should know the identifiers and data-names referenced in the LOCATION MODE clause since you may have to initialize them.

Location mode is particularly pertinent, then, under the following conditions:

1. When you are using the STORE statement.
2. When you are using a form of the FIND statement. (Record selection expression 5 can be applied only to a record having a location mode of CALC.) Refer to Section 2.2.4 for a discussion of record-selection expressions to be used with the FIND statement.
3. When the set occurrence selection mechanism is LOCATION MODE OF OWNER. (Before execution of a statement involving set occurrence selection, you must initialize any data-names and identifiers specified in and implied by LOCATION MODE OF OWNER.) Refer to Section 2.1.5 for a description of set occurrence selection.

One of three location modes must be specified for each record entry appearing in the schema. These are DIRECT, CALCulation, and VIA set name.

2.1.1.1 DIRECT – Storage (and retrieval) of an occurrence of the record type is based on the value in the DIRECT identifier; this value can be a data-base key (assigned by the DBCS to each record occurrence in the data base) or zero. If the value is the data-base key, DBCS uses the page number portion of the data-base key to locate the appropriate page. If the value is zero, DBCS uses the page number of the current record of the area to locate the appropriate page. If there is no current record of the area, DBCS uses the first page of the area's page range.

2.1.1.2 CALCulation – Storage (and retrieval) is based on the values supplied by the run-unit for the data-names contained in the specified record and declared as CALC keys. The DBCS transforms the values so provided into a unique identifier and stores the record on the basis of that identifier.

2.1.1.3 VIA Set Name – Storage (and retrieval) depends on the relationship established for the specified record by the DBCS on the basis of a set declaration in the schema. In effect, DBCS stores the record as physically close as possible to the logical insert point in the current set occurrence of the set type named in the VIA phrase. Its primary purpose therefore is to group related records together to minimize I/O accesses during retrieval.

2.1.2 Set Mode

Set mode – specified by the DBA using the schema DDL – describes the way in which the records in the set are related. Refer also to Section 1.4.2, which discusses (and illustrates) set and record occurrences. Currently, the only set mode in DBMS is CHAIN. In this mode, an embedded chain of pointers provides serial access to all records in that chain. The pointer is used to link one record in the chain to the next record in the chain. Chains can be processed either forward or backward from the current record. Normally, they are processed only in the forward (NEXT) direction unless the optional LINKED TO PRIOR clause is used. This clause causes the DBMS to facilitate processing in the backward (PRIOR) direction by maintaining a prior pointer for each record occurrence.

An additional optional clause, LINKED TO OWNER, can be specified for individual member-record types. This causes the owner record of the set to be accessible directly from each of the member records that have this clause specified. Set mode is completely transparent to an application program except in terms of execution-time efficiency. Refer to Section 2.5 for a discussion of efficiency considerations.

2.1.3 Set Order

The set order – specified by the DBA – controls the logical order of the member records within each set. The member records in a set may be ordered in one of the following ways.

1. SORTED in ascending or descending sequence based on the values of specified keys. The keys specified may be data-items in each of the member records, the names of the member records, or their data-base keys, or a combination of these.
2. In the order resulting from inserting new member-record occurrences into the set.
   – FIRST, that is, as the immediate successor to the owner record occurrence.

- LAST, that is, as the immediate predecessor to the owner record occurrence.
- NEXT, PRIOR, that is, after or before another record occurrence that is selected by the application program storing or inserting the record in the set.

2.1.4 Set Membership

Each set type must have an owner-record type and one or more member-record types. The DBA describes the owner of the set and its members in the schema. He also describes the way in which each member record participates in the set; i.e., its set membership. Set membership is either AUTOMATIC or MANUAL and either MANDATORY or OPTIONAL.

2.1.4.1 Automatic Set Membership – AUTOMATIC means that membership in the set is established by DBMS when a record occurrence is stored. That is, whenever an occurrence of a record declared to be an automatic member of a set is added to the data base, it will be logically inserted into (made a member of) the appropriate occurrences of all the sets in which it has been declared as an automatic member.

2.1.4.2 Manual Set Membership – MANUAL means that membership in the set can be established by a run-unit only by means of an INSERT command. The addition to the data base of a record occurrence declared to be a MANUAL member of a set will not cause it to be made a member of any occurrence of the sets in which it has been declared as a manual member.

2.1.4.3 Mandatory Set Membership – MANDATORY means that, once the membership of a record occurrence in a set is established, either automatically or by means of an INSERT command, the membership is permanent (as long as the set occurrence exists or the record is not deleted).

2.1.4.4 Optional Set Membership – OPTIONAL means that the membership of a record occurrence in a set is not necessarily permanent. Its membership can be cancelled by a REMOVE command or by a DELETE ONLY of its owner.

2.1.5 Set Occurrence Selection

Each time a particular set type is referenced implicitly during a STORE command, DBCS must resolve which occurrence of that set type to select to correspond to your statement. The Data Base Administrator can use either of two phrases to specify the set occurrence selection mechanism for a set in a schema: CURRENT OF SET or LOCATION MODE OF OWNER. You should be aware, however, that set occurrence selection is not applicable to sets whose owner is system. Since these constitute singular sets, DBCS can always correctly locate the set occurrence.

2.1.5.1 Current Of Set – If CURRENT OF SET is used, the DBCS stores the record in the set occurrence which has been accessed most recently by this run-unit (current of set). DBCS uses the last record accessed by this run-unit (current of record) as a reference point.

2.1.5.2 Location Mode Of Owner – If LOCATION MODE OF OWNER is used, DBCS selects the set occurrence by first locating an owner-record occurrence according to the location mode of the owner record as specified in the schema. If the location mode of the owner is DIRECT, DBCS uses the data-base key in the DIRECT phrase to locate the owner of the set. If the location mode of the owner is CALC, DBCS uses the CALC key to locate the owner of the set. You must, therefore, initialize the DIRECT key or the CALC key of the owner before referencing such a set type.

If the location mode of the owner is VIA set name, DBCS locates the owner of the (VIA) set in which the specified owner is a member. If the new owner also has a location mode of VIA, DBCS repeats the process until it finds a set occurrence selection of CURRENT OF SET, the SYSTEM record, or an owner with a location mode of DIRECT or CALC. DBCS then calculates the set occurrence back down the hierarchy until it reaches the owner originally specified. DBCS then stores the new record in that set occurrence.

2.2 WRITING DML STATEMENTS IN AN APPLICATION PROGRAM

To access the data base, you must include Data Manipulation Language (DML) statements among the host language statements in your program. As noted, first carefully examine the schema. Familiarize yourself with the DML statements in Chapter 3 and the conventions for their use. Also be sure to note the conventions for using the DML within COBOL or FORTRAN. (Refer to Chapters 4 and 5 for the specific usage of the DML within COBOL and FORTRAN, respectively.) The following sections introduce the DML statements and generally discuss their use.

2.2.1 Invoking a Sub-Schema

The first DML statement to include in a program-unit is INVOKE (or ACCESS, refer to 2.2.2). The INVOKE statement specifies the sub-schema that the program-unit will reference.

Each INVOKE statement causes the COBOL compiler or the FORTRAN preprocessor (FORDML) to create a User Working Area (UWA). (Refer to Section 1.5.3 for a description of the UWA.) Each data-item included in the sub-schema is assigned a location in the UWA and can be referenced by its name as declared in the schema. You cannot reference data-items in the schema that are not included in the sub-schema you invoke.

The data descriptions from the schema are placed in your application program in the form of the record descriptions used by the host language. The system communications area (that is, the special registers) are also placed in the UWA. These registers are used to store the error status; the names of the area, set, and record where the error occurred; and the last record and area affected by the MOVE statement. Refer to Chapters 4 and 5 for the descriptions of these registers as they are declared for COBOL and FORTRAN.

The Data Base Administrator can assign a privacy lock to any sub-schema. A privacy lock is a single alphanumeric value at most five characters in length. When a sub-schema has a privacy lock, include a privacy key in the INVOKE statement. If you specify a privacy key longer than five characters, it will be accepted and truncated. Refer also to Section 1.5.4 for a discussion of protection of data.

Only one INVOKE statement can be present in a program-unit since you can reference only one sub-schema in a program-unit. Within a run-unit, however, you can reference up to eight sub-schemas. The name of each sub-schema must be unique. In general, note that schema names, sub-schema names, and privacy-lock names used with an INVOKE or ACCESS statement effectively constitute user-reserved words. They must be unique. You cannot use them in other parts of your application program to name other items.

When using more than one INVOKE statement in a run-unit, you must inform the DBCS which sub-schema is current so that the DBCS can reference the correct sub-schema. Do this by calling the DBMS SETDB and UNSET subprograms.

The SETDB subprogram sets the current sub-schema. The sub-schema must have been previously invoked. Note that an INVOKE statement implicitly calls SETDB.

For COBOL programs, the form of the call to SETDB is:

ENTER MACRO SETDB USING 'sub-schema name'.

Note that the upper-case characters with underscoring indicate keywords from the DML that must be used when the formats of which they are a part are used. Refer to Section 3.1 for an explanation of the conventions used in descriptions of DML statements.

For FORTRAN programs, the form of the call to SETDB is:

CALL SETDB ('sub-schema name')

The conventions referred to in the COBOL-DML example also apply to the FORTRAN-DML example.

The UNSET subprogram causes DBCS to make the sub-schema that was previously current to be current again. That is, each call to UNSET causes a sub-schema to be removed from a stack of sub-schemas that was loaded by calls to SETDB.

For COBOL programs, the form of the call to UNSET is:

```
ENTER MACRO UNSET
```

For FORTRAN programs, the form of the call to UNSET is:

```
CALL UNSET
```

Note again that an INVOKE statement implies a call to SETDB. Therefore, include a call to UNSET for each implicit or explicit call to SETDB in any one of the following cases.

1. If you use more than one INVOKE statement in a run-unit.
2. If a single INVOKE statement is processed more than once in a run-unit.
3. If one or more calls to SETDB occur in a run-unit.

The only call to SETDB that does not require a corresponding call to UNSET is a single INVOKE statement processed once in a run-unit.

A subprogram that contains a main entry point with an INVOKE statement and one or more secondary entry points must be called the first time through the main entry point because it contains the INVOKE statement. The first call to this entry point accomplishes the runtime binding to the data base and sets the sub-schema current. (Should an exception occur during binding, it would be associated with the BIND statement code. Refer to Section 3.2.2.) Subsequent calls to this entry point will only set the sub-schema current. When another entry point is called, include an explicit call to SETDB at the entry point so that the sub-schema-setting function of the INVOKE statement will be accomplished. Then include a call to UNSET before making the return to the calling program. Also, if there are multiple entry points in the subprogram containing calls to SETDB, do not pass subprogram flow through more than one of these entry points (including the main point that contains the INVOKE statement). Otherwise, you must make a corresponding number of calls to UNSET before the return to the calling program.

The placement of the INVOKE statement differs for each of the host languages. Refer to Chapter 4 for COBOL and Chapter 5 for FORTRAN. The format and rules for the INVOKE statement are given in Chapter 3.

2.2.2 Accessing a Sub-Schema Invoked in another Program-unit

The ACCESS statement enables a subprogram to access the sub-schema invoked in another program-unit (the main program or another subprogram). When an ACCESS statement is processed, the User Working Area (UWA) of the calling program-unit is made available to the called subprogram. The way in which this occurs and the placement of the ACCESS statement depend on the host language used. (Refer to Chapter 4 for COBOL usage of ACCESS and Chapter 5 for FORTRAN usage of ACCESS.) Once the UWA is made available to the subprogram, the sub-schema can be accessed by the DML statements in the subprogram as it would be accessed in the calling program.

You can mix ACCESS statements in a run-unit with INVOKE statements as long as you include only one of either statement in a single program-unit. An ACCESS statement does not imply a call to either of the subprograms SETDB or UNSET. (These DBMS subprograms are described in Section 2.2.1 with the INVOKE statement.) If a subprogram is always called by a program-unit that has invoked the sub-schema referenced by the ACCESS statement, do not include calls to SETDB or UNSET in that subprogram. However, if a subprogram containing an ACCESS statement can be called by any program-units that do not reference the same sub-schema as that referenced by the ACCESS statement, include explicit calls in the subprogram containing the ACCESS statement to:

1. SETDB to set the sub-schema current, and
2. UNSET before making the return to the calling program.

Refer to Chapter 3 for the format and rules for the ACCESS statement.

2.2.3 Opening Areas

The OPEN statement opens one or more areas in a data base for your use. You can request that one or more specific areas be opened or that all areas included in the sub-schema be opened. With the OPEN statement you must also indicate the usage mode describing how the area is to be accessed. The usage modes are

- UPDATE
- RETRIEVAL
- EXCLUSIVE UPDATE
- EXCLUSIVE RETRIEVAL
- PROTECTED UPDATE
- PROTECTED RETRIEVAL

Concurrent run-units cannot gain access to the area over which an already active run-unit has EXCLUSIVE rights. Concurrent run-units can retrieve but cannot update an area for which an already active run-unit has PROTECTED rights. UPDATE (without EXCLUSIVE or PROTECTED) allows concurrent run-units with UPDATE or RETRIEVAL usage modes to open the same area. RETRIEVAL (without EXCLUSIVE or PROTECTED) allows concurrent run-units with UPDATE, RETRIEVAL, PROTECTED UPDATE, and PROTECTED RETRIEVAL usage modes to open the same area. Refer to Table 3-2 (in Chapter 3) for more details on usage-mode conflicts.

At the schema and sub-schema levels, areas can be designated to be schema temporary or sub-schema temporary. When you open an area designated temporary, you are allocated a personal copy of that area. You can modify the data in your copy as if the data were open in EXCLUSIVE-UPDATE usage-mode. This can be particularly useful when you are testing programs with 'live' data since you will not interfere with the integrity of the data base. The data base area, itself, is treated as if you had opened it in PROTECTED RETRIEVAL usage-mode; that is, concurrent run-units are allowed to retrieve while you are using your personal temporary copy. When you close the area, any changes you have made are discarded. DBCS then makes the area fully available for use by other run-units. Note that during this time – that is, while you have a specified area open for temporary use – you can open other areas of the data base in any of the six usage-modes.

An area (as well as a sub-schema) can have a privacy lock. With the OPEN statement you must supply the appropriate privacy key to be able to access the area.

Note also that when the owner of a set is SYSTEM (constituting a singular set) the area that incudes the system set must be in each sub-schema in which that set is included. You need open it, however, only if you reference the system record.

2.2.3.1 Opening Areas Simultaneously with Other Run-Units – As noted in the description of usage-modes, DBMS allows you to update or retrieve data while another run-unit updates or retrieves data in the same area. This facility has been termed simultaneous-update. Strictly speaking, however, the name is misleading since the facility includes a retrieving usage-mode. The simultaneous-update facility operates when a run-unit opens an area in one of three usage-modes:

UPDATE
PROTECTED UPDATE, and
RETRIEVAL.

Concurrency is maintained through use of the ENQUEUE/DEQUEUE facility of the DECsystem-10 operating system. (Refer to Chapter 16 of the *DECsystem-10 Monitor Calls Manual* for a discussion of ENQUEUE/DEQUEUE.)

When deciding how to open an area (that is, which usage-mode to specify), keep in mind that (1) a significant aspect of using an area in UPDATE usage-mode is that the journal file must then be shared with other run-units; and (2) the ENQUEUE and DEQUEUE that occurs within a command or a transaction is comparable to using a simple DML command (for example, FIND NEXT) in terms of CPU usage.

In general then, if you have opened an area in a simultaneous-update usage mode, and are executing updating commands, each command will potentially be locking out other run-units from the data base resource. This decreases the throughput that DBCS is capable of (relative to the other usage modes). If you are in an on-line environment, in particular, it is important to consider this impact on throughput when you are designing your updating algorithms. Because of this, it is important not to use simultaneous-update unless you really need its functionality. Table 2-1 lists all six usage modes and gives some suggestions for using each advantageously. (See Table 2-2 in the *Administrator's Procedures Manual* for efficiency and design considerations in using simultaneous update.)

Table 2-1 Usage Modes with OPEN; Suggestions for Efficient Use

RETRIEVAL*	You intend other run-units to open simultaneously with UPDATE or PROTECTED UPDATE.
UPDATE*	You intend other run-units to open simultaneously with UPDATE.
PROTECTED RETRIEVAL	You expect concurrent retrievers but no concurrent updaters.
PROTECTED UPDATE*	You intend other run-units to open with RETRIEVAL.
EXCLUSIVE RETRIEVAL	You really need this exclusiveness; cost is equivalent to PROTECTED RETRIEVAL.
EXCLUSIVE UPDATE	You really need this exclusiveness; this is the minimum cost update usage-mode.

* Simultaneous-update usage-mode.

Within the framework of simultaneous update then, use of the data-base resource is either exclusive or shared. Exclusive use occurs during execution of updating commands. For the duration of any updating command, no other run-unit can access the data base since lock-out is maintained at the data-base level.[1] Consider the following example. Run-unit A opens an area for update. If during execution of a DML updating command issued by Run-Unit A, Run-Units B and/or C attempt to initiate a data-base accessing command, each must wait in a queue until Run-Unit A's updating command is completed. They have no indication from the operating system, however, that they are waiting. They are placed in the queue in the order in which their requests are received – and they are serviced in that order.

If updating commands are not being executed, then use of the data base resource is shared. For example, if Run-Unit A executes a FIND command, then Run-Units B and C can also simultaneously execute FIND, GET, and IF commands. If one of these run-units wants to update, however, it must wait until the retrieval commands of the others have been completed. It then locks the data base resource exclusively for the duration of its commands.

2.2.3.2 Using Areas Simultaneously with Other Run-Units – DBCS cannot determine which areas a DML command will access before this access occurs. Because of this, a run-unit is considered to be within the domain of the simultaneous-update facility if the run-unit has at least one area open in a simultaneous-update usage-mode.

[1] The duration for which a run-unit retains the data base exclusively is termed an interleaving unit.

Should you decide to open a data-base area in one of the three usage-modes that allow simultaneous access, be sure to note the form of the IMAGES statement in the schema you are referencing. (See Figure 2-1 for an illustration of a schema.) The DMCL portion of the schema contains the IMAGES statement.

If IMAGES are in order by command, DBCS retains the data base exclusively for the duration of each DML updating command you issue and retains it shared for FIND, GET, and IF commands. (If you wish, you can also define transactions using JSTRAN–JETRAN; you may wish to do this to organize your program or to control your program in a specific way.)

If IMAGES are not in order by command, you can define the duration for which DBCS retains the data base exclusively. You do this by defining transactions using JSTRAN–JETRAN. (Refer to Section 2.3.5.1 for details on using these subprograms.) The data base is then retained exclusively for the duration of the transaction – from the issue of your call to JSTRAN to the issue of your call to JETRAN. In addition, when IMAGES are not in order by command, any DML commands issued outside a defined transaction are treated by the rules applicable to IMAGES by command. You must not, however

- issue a JSTRAN while you have another transaction active. (You would then be issuing two successive JSTRANs without issuing a JETRAN in between.)
- perform a COBOL RETAIN during a DBMS transaction.
- use JBTRAN with other than a 0 argument (see Section 2.3.2, which describes JBTRAN use) when you are within the simultaneous-update domain. This means you are allowed to restore the data base to the beginning of the most recent transaction.

If you do any of the above, you will get an exception. See Appendix B, Table B-2 for a description of exception conditions.

2.2.4 Walking through Structured Data

After opening an area, you can select a particular record for processing by using the FIND statement. With the FIND statement, you must indicate which method of record selection you want to use. Five are available; in effect, they are the search arguments used for selecting records from the data base. They are called record-selection-expressions (rse) and can be classified as follows:

1. Selection of a record by means of its data-base key (rse 1)
2. Selection of a record by means of currency indicators (rse 2)
3. Selection of a record by means of its relative position in a set or area (e.g., NEXT, PRIOR) (rse 3)
4. Selection of the owner record of a set (rse 4)
5. Selection of a record if its LOCATION MODE is CALC (rse 5).

For example:

```
FIND REC2 USING KEY1.                    (rse 1)
FIND NEXT REC2 RECORD OF SETA SET.       (rse 3)
```

In some cases the record occurrence you want to access must be found by means of other records or sets. That is, first another record or set must be found. Then the logical relationships established for the set in the schema must be followed until a record that participates in more than one set is found. This, in effect, is a junction. A branch can then be made to another set and followed until the record occurrence desired is found, or until another junction.

Figure 2-3 illustrates this procedure. To find the tenth record occurrence of record type REC6, for example, when the current record is an occurrence of REC2, first find the owner of the current set (SET0), which is also the owner of SET1. Then find the member of SET1 (REC3). Continue finding owner and member records until you reach REC6. Then find the tenth occurrence of REC6. To successfully follow this procedure, check the schema and sub-schema for information about the sets to search. (Section 2.1 describes the schema and sub-schema.)

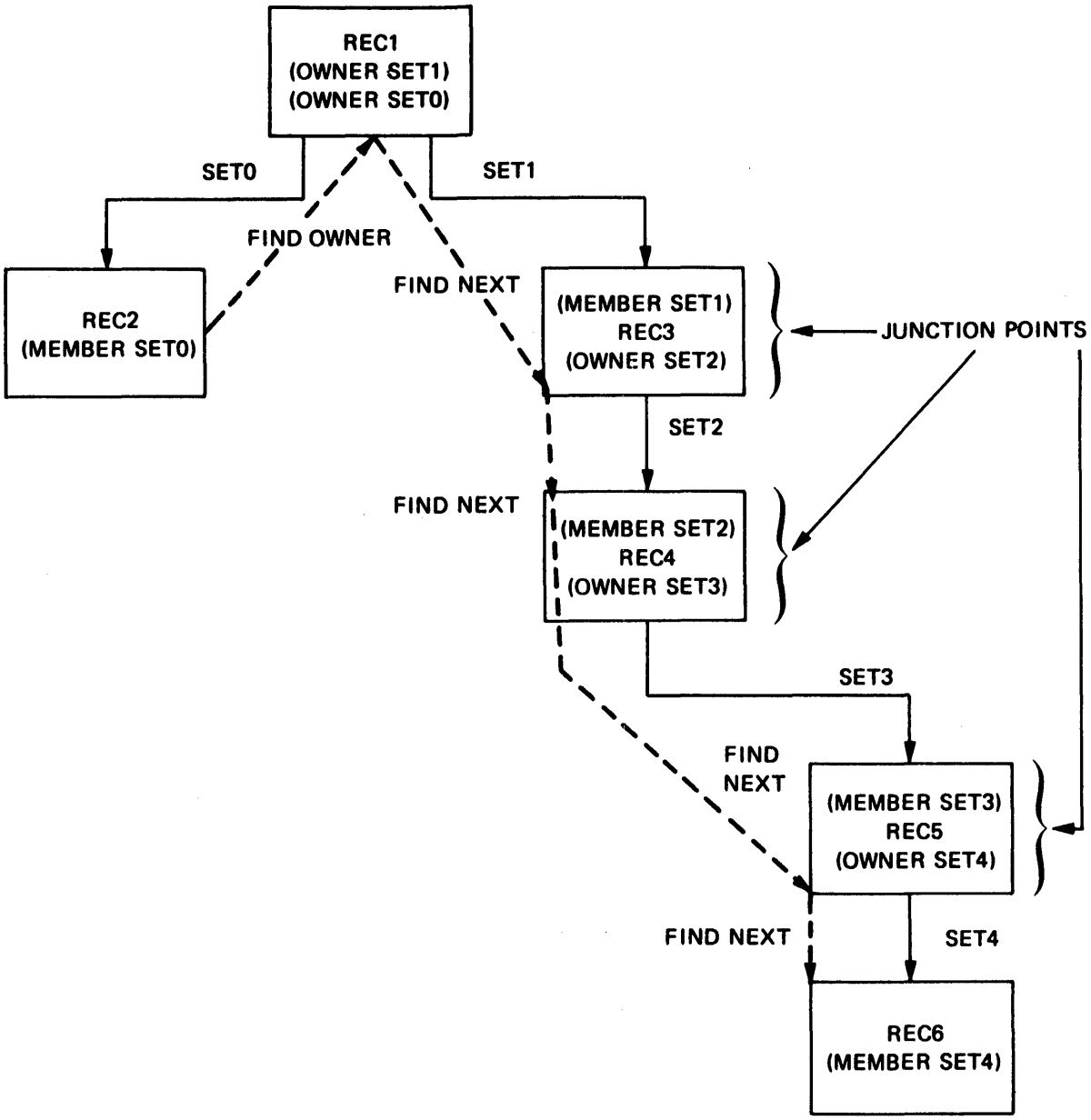


Figure 2-3 Walking through Structured Data

2.2.5 Retrieving Data

To transfer records or data-items within records to the UWA from the data base, use the GET statement. The record that is to be transferred is always the current record of the run-unit (that is, the last record accessed by that run-unit). Refer to Chapter 3 for the format and rules for the GET statement.

The MOVE STATUS statement allows you to save the contents of a currency status indicator. This is particularly useful when you want to access a record in another part of the data base without losing your place in the part of the data base you are currently using. MOVE STATUS also alters the special registers AREA-NAME and RECORD-NAME to describe the currency indicator being moved.

2.2.5.1 Currency Status Indicators – The currency status indicators are single-word registers that record the data-base key of the:

1. last record accessed by the run-unit – CURRENT of RUN-UNIT
2. last record accessed of each record type of the sub-schema - CURRENT of RECORD
3. last record accessed in each set of the sub-schema - CURRENT of SET
4. last record accessed in each area of the sub-schema - CURRENT of AREA

When CURRENT of SET has been deleted or removed – and a new CURRENT OF SET has not been established determination of NEXT OF SET (with FIND) is as follows. The record which was NEXT of the eliminated record becomes NEXT OF SET. The record PRIOR to it at any point in time becomes PRIOR of SET. Deleting the CURRENT of AREA, however, has no effect on the meaning of NEXT or PRIOR of AREA.

Currency status indicators are, in effect, place markers kept by the DBCS for a run-unit. At the start of execution of a run-unit, the currency status indicators are null. You can selectively suppress the updating of the currency status indicators, except for the current of run-unit, by including the SUPPRESS phrase (in the FIND statement) in your program.

Refer to Chapter 3 for the format and rules for the MOVE STATUS statement.

2.2.6 Performing Updates

Five statements constitute the update class of statements. These are STORE, MODIFY, INSERT, REMOVE, and DELETE.

To create a new record in the data base, use the STORE statement. Both the physical location of the record and the implied set linkages are determined by DBCS from the information supplied in the schema. Consequently you do not specify either. In some cases, however, neither the locations nor the linkages are completely transparent. Refer to Section 2.1.5 on set occurrence selection for more information.

To change values within a record, use the MODIFY statement. It always modifies the record that is the current of run-unit (that is, the last record accessed).

To add a record to a set or to remove it from a set, use the INSERT and REMOVE statements, respectively. To use these statements properly, you should know the way in which the record participates as a member in the set. A record can be an AUTOMATIC or MANUAL and a MANDATORY or OPTIONAL member of a set. (The Data Base Administrator determines a record's membership. You can find this information in the schema.) AUTOMATIC records are linked to the sets in which they are members automatically (that is by DBCS) when you store them. You must link MANUAL records to the sets by using the INSERT statement. Further, you can disconnect records from sets in which they are OPTIONAL members by using the REMOVE statement. You cannot affect the membership in a set of MANDATORY records with REMOVE. You can, however, change MANDATORY records with the MODIFY statement.

To eliminate a record from the data base, use the DELETE statement. Since a DELETE statement can cause the deletion of many records, it is important to fully understand its operation. Confer closely, therefore, with the Data Base Administrator to ensure that no records are unintentionally deleted.

Refer to Chapter 3 for the formats and rules for the STORE, MODIFY, INSERT, REMOVE, and DELETE statements.

2.2.7 Closing Data Areas

When you have finished accessing the data in the area of the data base that you have opened, close the area using the DML CLOSE statement. When a run-unit closes an area, that run-unit cannot access that area unless the run-unit again opens that area. Note that the CLOSE statement also has implications for the journal file. The format of the CLOSE statement you use, for example, can cause DBCS (1) to append to the journal file next time the file is opened, or (2) to possibly overwrite the contents of the journal file next time it is opened. Refer to Chapter 3 for the formats and rules for the CLOSE statement.

2.3 CREATING A JOURNAL FOR BACKUP AND RECOVERY

When you open areas in the data base in UPDATE, or EXCLUSIVE/PROTECTED UPDATE usage modes, the data in these areas is vulnerable to erroneous changes or system crashes. The DBA typically specifies a journal file (in the DMCL of the schema) to protect the data base. If a journal file has not been specified, you can create a journal file during execution of your application program. The journal file can be used in conjunction with the DBMEND utility (refer to Chapter 6 of the *Data Base Administrator's Manual*) to recover the data base if problems occur.

The journal file can contain BEFORE/AFTER images of those parts of the data base that are being changed by updating run-units. BEFORE images are copies of pages of the data base as they were before changes were made. AFTER images are copies of the pages as they are after changes are made. BEFORE images are used to return the data base to an earlier state because a run-unit had errors or terminated abnormally. AFTER images are used to place into the data base those changes that are known to be correct, but that have not yet been made to the data base.

If the DBA has specified a journal file, he has also specified the kind of images that can be written into the journal in the BACKUP clause of the DMCL. If necessary, he can specify both BEFORE and AFTER images. If the BACKUP clause has been omitted, a journal file is not created automatically when your run-unit opens an area for update. Should you need a journal file, you can call the subprogram(s) described in Section 2.3.3 and specify the kind of images you require.

Note that the journal file is the means DBMS-10 uses to provide recovery if an exception occurs during an updating command. To have this run-time recovery, however, you must have BEFORE images. If no journal file has been specified with BEFORE images, either through the DMCL BACKUP clause or through a run-time subprogram call, no run-time recovery is possible. Note also that recovery with DBMS-10 can be incremental. This means that the amount (or increment) of backup/recovery can be controlled by means of the IMAGES statement of the DMCL. Either commands or transactions can be specified.

When BEFORE images have been specified and images by command have been specified, DBCS writes a command header and BEFORE images into the journal file as a DML command updates the data base. When the DML command is successfully completed, DBCS writes the command trailer. If an exception occurs during an updating command, DBCS automatically restores the data base back to the last command header in the journal. When images by command have not been specified, you can perform recovery only in increments of transactions. You do this by calling JSTRAN–JETRAN to write the transaction headers and trailers. When an exception occurs, you can then call the JBTRAN subprogram to restore the data base back to a specified transaction header. Note again that both types of recovery require BEFORE images.

The remainder of this section describes different aspects of journaling as follows:

- Section 2.3.1 describes journaling within simultaneous update.
- Section 2.3.2 describes journaling by command and by transaction.
- Section 2.3.3 describes how you can specify a journal file.
- Section 2.3.4 describes how to assign the journal file to a device.
- Section 2.3.5 describes the types of information in the journal file and the subprograms you can call to add each type.

2.3.1 Journaling within Simultaneous Update

When two run-units update the same area simultaneously, they also update the journal file simultaneously. This simultaneous use of the journal file forces DBCS to do some extra work to ensure the continuing integrity of the journal file. It is important, therefore, that you inform DBCS when you do not need to share the journal file. You do this by including the OPEN JOURNAL USAGE-MODE EXCLUSIVE UPDATE statement in your program before opening any data base areas. Refer to Section 3.3 for the discussion of formats and rules of the OPEN JOURNAL statement.

Upon opening the journal file, you will receive a message from DBCS describing the journal-file characteristics. The form of this message is:

```
[JOURNAL CHARACTERISTICS:]

RUN UNIT ID    id value
schema-name    RUN schema-run-of-last-overwriter
date/time-journal-last-overwritten
```

The message above shows the RUN-UNIT ID; it is the identification DBCS gives your run-unit. This identification is given so that changes made by your run-unit can be isolated when DBMEND is used to recover the data base.

If the id value is 0, you are the first overwriter of the journal file. This means that DBCS had determined that the information (previously existing) in the journal file was no longer needed – and had marked the journal file's label page to so indicate. When your run-unit opened the journal, therefore, it went to the beginning and wrote over pre-existing information. If the id value is not 0, you are appending to the journal file. Note that the RUN-UNIT ID is returned to 0 each time the journal file is overwritten.

The schema-run-of-last-overwriter indicates the state of the schema file at the time the journal file was last started anew (overwritten).

Refer also to Section 2.2.3.1 for a discussion of simultaneous update.

2.3.2 Journaling by Command and by Transaction

Using the IMAGES statement, the DBA can specify the degree – or increment – of journal recoverability for each data base. Two possibilities exist:

1. Images ordered by command
2. Images ordered by transaction.

2.3.2.1 Images Ordered by Command – The default is for images to be ordered by command. When images are ordered by command, the pages changed by a DML updating verb are always forced out to the journal file and to the data base at the completion of each command. (The updating verbs are STORE, MODIFY, INSERT, REMOVE, and DELETE.) This means more writing is done to the data base and to the journal. (See Section 2.5, which discusses efficiency considerations.) It also means that the data base can be backed up in increments of single updating verbs. Should an exception occur under these circumstances, the DBCS will automatically restore the data base to the state it was in before the verb was entered – if BEFORE images have been specified. In a shared environment (simultaneous update) your run-unit retains the data base for the duration of an updating command.

2.3.2.2 Images Ordered by Transaction – When images are not by command, you can specify that images be ordered by transaction. Force-out then occurs only after each transaction you specify – using JSTRAN and JETRAN. Less writing is done to the data base and to the journal file in this case, but the data base can be recovered only in increments of transactions; and in a shared environment (simultaneous update) your run-unit retains the data base for the duration of a transaction.

If BEFORE images have been specified, you can restore images by transaction by calling the JBTRAN subprogram. You may want to do this for one of two reasons:

1. to restore the data base to the beginning of a transaction in which an exception has occurred, or
2. to erase one or more transactions (for reasons specific to your application).

To use JBTRAN to restore the data base to the beginning of the most recent transaction, the form of the calls are as follows:

For COBOL:

```
ENTER MACRO JBTRAN USING 0
```

For FORTRAN:

```
CALL JBTRAN (0)
```

Note that if you are within simultaneous update you can use JBTRAN only with a 0 argument.

To use JBTRAN to erase one or more transactions, give the number of transactions you want erased to JBTRAN as an argument. The form of the calls are as follows:

For COBOL:

```
ENTER MACRO JBTRAN USING { integer                }
                         { variable USAGE comp    }
```

For FORTRAN:

```
CALL JBTRAN (integer)
```

2.3.3 Specifying a Journal File

A journal file is created automatically if a program opens an area in UPDATE, or EXCLUSIVE or PROTECTED UPDATE mode when the schema declaration for that area contains a BACKUP clause. This clause specifies that BEFORE and/or AFTER images will be placed in the journal file. If the BACKUP clause is not present in the schema declaration for the area, and you want to have a journal file created (i.e., have BEFORE or AFTER images for an area in the journal file), include a call to one of the JMxxx subprograms specifying the name of the area. You can also use a call to one of these subprograms to override the BACKUP clause in a schema declaration. This will not change the BACKUP clause in the schema declaration, only the kind of image actually used in the particular run-unit calling the subprogram.

For COBOL the form of the calls to the JMxxx subprograms is as follows:

```
ENTER MACRO { JMAFT  }  [ USING { identifier-1 } ]
            { JMBEF  }          { literal-1    }
            { JMBOTH }
            { JMNONE }
```

The value of identifier-1 or literal-1 is the name of the area that will have AFTER, BEFORE, both, or no images included in the journal file. If no USING phrase is specified, all areas in the program are affected.

For FORTRAN the form of the JMxxx calls is as follows:

```
CALL { JMAFT  }  [ (string) ]
     { JMBEF  }
     { JMBOTH }
     { JMNONE }
```

The value of the string is the name of the area that will have AFTER, BEFORE, both, or no images included in the journal file. Refer to Appendix D for a discussion of string arguments in FORTRAN. If the string is not specified, all areas in the program are affected.

If you make a call to one of the JMxxx subprograms, do it before opening any areas.

2.3.4 Assigning the Journal File to a Device

Using the DMCL JOURNAL statement, the DBA can describe the journal file specification. If the journal file specification is not given in the schema, note that the default specification of the journal file is

JRN:schema-name.JRN

That is, the device-name is JRN; the filename is the name of the schema being used; and the file extension is .JRN.

The journal file can reside on magnetic tape or on disk. If the DBA has not made actual device assignments in the journal file specification, you must do so. Before running your application program, you can assign the journal file to a device with the ASSIGN monitor command. You would assign the journal file to a magnetic-tape device as follows:

.ASSIGN MTAn:JRN↵

Refer to the *DECsystem-10 Operating System Commands Manual* for a complete description of this command. Refer also to Section 6.3 in the *Administrator's Procedures Manual.* This section discusses the DAEMDB program, which can be used to perform magnetic-tape journaling.

Although it is not recommended, you can change the name of the journal file from that specified in the schema. To do so, use JMNAME. You would use JMNAME to give the journal file a filename other than that of the schema or to permanently specify an actual device name. You can also use JMNAME to specify a directory for the file. Call the JMNAME subprogram before opening any areas.

Note again it is not recommended that you change the filename from that specified in the schema. Should you choose to do so, you are responsible for using a valid file specification. The system does not check the values. You will not be aware, therefore, that you have an invalid specification until DBCS attempts to open the journal.

For COBOL the form of the call to JMNAME is:

ENTER MACRO JMNAME USING { identifier-1 / literal-1 }

The value of identifier-1 or literal-1 is the file specification. The file type must be specified as .JRN.

For FORTRAN the form of the call to JMNAME is:

CALL JMNAME (string)

The value of the string is the file specification. The file type must be specified as .JRN. Refer to Appendix D for a discussion of string arguments in FORTRAN.

2.3.5 Information in the Journal File

The main contents of journal files are the BEFORE and AFTER images of those pages of the data base that are modified during the execution of the application program. In addition, the first page of each reel of the journal file is a label block. It contains the schema name, a run-number, a reel number, and the date and time the run began. When images are ordered by command – or when commands are outside the context of a transaction in a run-unit using simultaneous update – each set of pages changed by a DML statement is delimited by a command header and trailer. The header and trailer gives the name of the DML command that caused the pages to be changed and a command index. The label blocks, command headers/trailers (when applicable) and pages from the data base are all automatically generated by the DBCS while your application program is running. But if you want, you can add other forms of data to the journal file. The following sections describe the type of data you can add.

2.3.5.1 Adding Transaction Headers-Trailers (JSTRAN–JETRAN) – You can further define the activity of your application by adding transaction headers and trailers. To do so, call the DBMS subprograms JSTRAN and JETRAN, respectively.

For COBOL the form of the call for the JSTRAN subprogram is:

ENTER MACRO JSTRAN USING { identifier-1 / literal-1 } { identifier-2 / integer-1 }

The form of the call to the JETRAN subprogram for COBOL is:

ENTER MACRO JETRAN USING { identifier-1 / literal-1 } { identifier-2 / integer-1 }

The value of identifier-1 or literal-1 is the transaction name: this can be up to 30 alphanumeric characters. The value of identifier-2 or integer-1 is the transaction index; it must be specified as COMP single-precision.

For FORTRAN the calls for the JSTRAN and JETRAN subprograms are:

CALL JSTRAN (string, integer)
CALL JETRAN (string, integer)

The value of the string is the transaction name; it may contain up to 30 characters. The value of the integer is the transaction index. Call the JSTRAN subprogram immediately before execution of a group of DML commands that constitute a logical operation on the data base and modify the data base. The transaction header, containing the transaction name and index, is then placed in the journal file at the time of the call. Call the JETRAN subprogram after the group of DML commands has been executed. This will place the transaction trailer, also containing the transaction name and index, after the changes that occurred during the transaction. Ensure that the index is incremented after each pair of calls to these subprograms. The journal file will then contain the count of the number of times a transaction type has been processed.

2.3.5.2 Adding Comments (JRTEXT) – You can also add comments to the journal file by including a call to the JRTEXT subprogram in your application program.

For COBOL the form of the call is:

ENTER MACRO JRTEXT USING { identifier / literal }

The value of the identifier or the literal is a string of characters that represents the text of the comment.

For FORTRAN the form of the call to JRTEXT is:

CALL JRTEXT (string)

String is a string of characters that represents the text of the comment. Refer to Appendix D for a discussion of string arguments in FORTRAN.

2.3.5.3 Adding Nonprinting Data (JRDATA) – You can add nonprinting data to the journal file by calling the JRDATA subprogram. The data can be in any form because DBCS merely copies it into the file exactly as it is given.

For COBOL the form of the call to JRDATA is:

ENTER MACRO JRDATA USING { identifier / literal } , integer

The value of the identifier or literal is the data to be put in the journal file. The integer specifies a word count. If it is nonzero, the specified number of 36-bit computer words are copied into the journal file. If the first argument

specifies a COBOL string or group item, the word count can be 0 and the amount of data specified by the identifier or literal is copied into the journal file. Note that the data should start on a word boundary because DBCS copies starting from a word boundary.

For FORTRAN the form of the call is

CALL JRDATA ({ identifier / literal } ,integer)

The value of the identifier is the data you want to place in the journal file.

The literal is the first location of the data you want to place in the journal file. The integer specifies the number of 36-bit computer words of data that will be copied into the journal file. Note that the data should start on a word boundary because DBCS copies starting from a word boundary.

2.3.5.4 Adding Checkpoints (JRDATA) – Another function of the JRDATA subprogram is that it allows you to add checkpoints to the journal file so that your application program can be restarted after an abnormal termination. To add checkpoints, first determine the parameters of the job that you want to checkpoint (e.g., files that cannot be checkpointed with the RERUN program). For each transaction then, call the JRDATA subprogram giving the information to be checkpointed as the argument (e.g., the current block number in a file). Call JRDATA before calling JETRAN in this case. If the system crashes, you can create a checkpoint file from the journal file by means of the BUILD command of the DBMEND program. (Refer to Chapter 6 of the *Data Base Administrator's Procedures Manual* for information about DBMEND). You can then restart your application program by including code in your program to process this checkpoint file.

2.4 UNDERSTANDING DBCS CONTROL DURING PROGRAM EXECUTION

During the execution of an application program, DBCS provides two guarantees:

1. that a critical application is not returned to monitor level against its will, and
2. that a run-unit does not access the data base while the data base (or DBCS) is in an undefined state.

The following sections discuss some ways in which these guarantees can impact your interaction with the data base.

2.4.1 Program Return to Monitor Level

The assumption here is that a critical application may want to maintain complete control of the data base. The method to ensure this is to associate an exception code with all unusual/undesirable conditions. (See Section 3.2 for a discussion of exception handling and Table B-2 for a description of the exception-condition codes.) Therefore, DBCS need not exit the run-unit to monitor level without being specifically requested to do so – except in one case. This case is if the first call to DBCS is not the SBIND call generated by the INVOKE statement. If this occurs, it means that DBCS has not been successfully associated with a sub-schema. When this occurs, DBCS types

?DBSSNI SUB-SCHEMA NOT INITIALIZED YET

and exits to monitor level.

Typically, however, the INTERCEPT clause of the Device Media Control Language is used to request exits. The DBA can specify the class of exceptions that DBCS intercepts during a run-unit. If an exception of the specified class occurs, DBCS then types an error message and causes the run-unit to exit to monitor level. You can then decide what to do; among your options is typing the monitor command CONTINUE to continue execution.

2.4.2 DBCS or Data Base in Undefined State

This aspect of system control ensures that a run-unit does not access the data base while the data base or DBCS is in an undefined state. Some circumstances under which this can happen are as follows:

1. You are in command mode and journaling BEFORE images. In the process of executing an updating verb, an exception occurs. The system attempts automatic restoration of the data base to the beginning of the updating verb that caused the exception. If this attempt fails, you will get exception condition xx62 for each command you attempt thereafter indicating, in effect, an undefined data-base state. The only way to correct the situation is to stop program execution and use DBMEND to restore the data base.
2. You are in transaction mode and journaling BEFORE images. An exception occurs during a transaction. No automatic restoration is attempted by the system, and further execution results in exception condition xx62. To recover you must then use the JBTRAN subprogram with a 0 argument. (Refer to Section 2.3.1.2 for a discussion of JBTRAN.) If recovery is successful, the data base will be restored to the beginning of the unsuccessful transaction and you can continue execution. If recovery fails, you will get exception code 1661 indicating failure of the recovery process. You must then stop program execution and use DBMEND to restore the data base.
3. You are either in command or transaction mode, but are either not journaling at all or not journaling BEFORE images. An exception occurs during an updating verb and no automatic restoration is attempted by the system. All subsequent commands will return with exception condition xx62. You must then use an old copy of the data base for restoration.

Note that in all three cases the common circumstance is an undefined state either in the data base or in DBCS. Allowing a run-unit access to the data base during this state would endanger the integrity of the interface to other application programmers. For this reason, manual recovery is the recommended procedure.

2.5 EFFICIENCY CONSIDERATIONS

To run a program efficiently using DBMS, you should know the functions the system performs automatically during execution of a DML statement and be aware of the implications of this automation to your application. Because the DML provides a concise syntax for expressing common, but complex, computations, for example, its use can reduce development effort. This may have the effect, however, of increasing program execution time.

In general, you should understand the degree of control you can exert over specific aspects of system function to effect efficient use. Then you can decide which trade-offs are most important to your application and to your facility. This section therefore discusses three functions performed automatically by DBMS – and their implications. It also discusses journaling in the context of efficient use; and guidelines for using the DML efficiently.

2.5.1 Automatic Insertion of Records into Sets

Automatic insertion of records into sets may occur when you issue a STORE command. As described above, a STORE command is used to place new records in a data base. When using STORE, note that the record will be inserted in all sets in which it participates as an AUTOMATIC member. If the record is an AUTOMATIC member of many sets or a member of a large sorted set, hundreds or even thousands of record accesses could be implied by a single STORE command.

2.5.2 Implied Deletion of Records

Implied deletion of records can occur when a record occurrence to be deleted is the owner of a set. If deleted member records are owners of other sets, the members in those sets may also be deleted and so on through the data base. Be sure therefore that you are using the appropriate form of the command when you want to delete a record. Note also that many data base accesses can occur for a complex delete operation.

2.5.3 Maintenance of Sorted Sets

Maintenance of sorted sets means that DBCS must check the order of a sorted set each time you specify a record be stored in, deleted from, or modified in a sorted set. If the storage, deletion, or modification would cause the order to become unsorted, DBCS must reorder the set to maintain it sorted. Since a STORE statement, in particular, can affect many sets (as described above), maintaining sorted sets can also affect STORE execution time considerably.

2.5.4 Journaling

Another function that is an intrinsic part of DBMS is journaling. Using the IMAGES statement, the DBA can specify the increments in which updates are written out to the data base and to the journal. Two possibilities exist:

1. By command – journaling in increments of single updating verbs.
2. By transaction – journaling in increments of user-specified transactions (during a call to JETRAN–JSTRAN).

Each of these methods of journaling has an impact on efficiency. When images are ordered by command, updates are forced out to the journal and to the data base after execution of each DML updating verb. This means that more writing is done to the data base and to the journal file. It also means more control over the data base, however, since the data base can be restored in increments of single updating verbs – in DBMEND and at run-time. Journaling by command is the default.

When journaling is not by command, updates are forced out to the journal and to the data base only after each user-specified transaction. This means less writing is done to the data base and to the journal file; but, it also means less control over the data base since the data base can be restored only in increments of user transactions – again in DBMEND and at run-time. You can call the subprogram JBTRAN to specify run-time database restoration by transaction. (Refer to Section 2.3 for a discussion of JBTRAN.)

The decision as to which method to use is essentially a cost-benefit trade-off for the facility. In general, this type of decision is left to the DBA. Refer, however, to Section 2.3 for a detailed discussion of journaling and to Section 3.2 for a description of exception handling involving back-up of the data base.

2.5.5 Guidelines for Efficient Use of DML

The following are some guidelines for efficiently using the DML.

1. When the records in a set are frequently updated (particularly when they are being deleted or removed), define PRIOR pointers in addition to the always-present NEXT pointers. Doing so will improve execution efficiency.
2. When the member-record occurrences of a set are usually accessed serially (i.e., when a FIND rse 3 such as FIND NEXT is used), declare the LOCATION MODE of each member record as VIA in the schema so as to physically localize each set occurrence.
3. Unless sorted sets are necessary (e.g., needed for alphabetized reports that are printed relatively frequently), do not use them since the overhead they involve cannot be justified for sets with significant record activity.
4. Use CALC keys only when a record type must be accessed randomly. Otherwise, the overhead involved in maintaining CALCed record types is probably not justifiable.
5. Use localized logical accessing rather than nonlocalized accessing. For instance, when using two groups of records, access all of one group then all of the other group rather than going back and forth.
6. When using the MODIFY statement on a member of a sorted set, specify only the data-items to be modified. Otherwise, the DBCS (unnecessarily) re-inserts/sorts the record being modified in the set occurrence (since it has no other way of determining if the sort-keys have or have not been modified).
7. Avoid set occurrence selection using LOCATION MODE OF OWNER unless it is necessary because it can cause the DBCS to unnecessarily search the data base for the correct owner. For example, in this mode when storing two member records, one right after the other, into the same set occurrence, the DBCS would redundantly reselect the owner record during the second STORE operation.
8. Specify LINKED TO OWNER if set ORDER is FIRST or if SORTED, set occurrence selection is CURRENT, and the current of set was not determined relative to the owner.
9. Choose the degree of journaling appropriate to the application. (See also Section 2.3.)
10. Open the data base only for the access needed to accomplish the function of your application. Note that the overhead for using simultaneous update is high; do not therefore open the data base in the simultaneous-update usage-modes unless your application requires it. See also Section 2.2.3 on opening areas and Section 3.3 on specifying usage-modes with the OPEN statement.

11. The **CALC** algorithm tends to work best when a record's range is an odd number of pages. Therefore, specify the **FIRST PAGE/LAST PAGE** as both even or both odd when you use these phrases in the **DMCL**. See also page one of Figure 2-1, the sample schema and sub-schema listing.

CHAPTER 3
DATA MANIPULATION LANGUAGE

The Data Manipulation Language (DML) provides you with the capability to interact with a database. When your request for data in the database is successfully completed, the requested data is deposited in the User Working Area (UWA) of the calling program-unit; you can then reference and manipulate the data using the facilities of the host language. Should you want to add new data or return modified data to the data base, use the DML to request the appropriate action from the Data Base Control System (DBCS).

This chapter defines the specifications for the DML. Section 3.1 discusses the conventions used to write DML statements; Section 3.2 describes exception handling and the error special registers; Section 3.3 lists and describes the DML statements; and Section 3.4 describes the FORTRAN intrinsic functions provided for DBMS use.

3.1 DML STATEMENT CONVENTIONS

Certain conventions have been used in the descriptions of the DML statements in this chapter and generally throughout this manual. These conventions and their meanings are as follows.

Lower-case characters	Information that you must supply, such as values, names and other parameters.
Upper-case characters underscored	Key words in the DML lexicon you must use when using the formats of which they are a part.
Upper-case characters not underscored	Other words in the DML lexicon that serve only to make the DML statements more readable. Their use is optional and has no effect on the meaning of the formats of which they are a part.
Braces { }	A choice. Choose from the two or more lines enclosed.
Brackets [ ]	An optional feature. The contents of the brackets are used according to the rules above if you choose the feature.
Ellipsis ...	Repetition. The information contained within the preceding pair of braces or brackets can be repeated at your option.
Double Vertical Lines ‖ ‖	A choice. Choose one, several, all, or none of the lines enclosed.

Note that the semicolon (;) and comma (,) are treated as spaces in all DML statements. The only punctuation required in these statements is a period to end the statement. In the examples in this manual, a semicolon or comma is used for readability only and does not affect the meaning of the statement.

3.2 EXCEPTION HANDLING

Using the INTERCEPT and NOTE clauses of the Device Media Control Language, the DBA can specify the class of exceptions that DBCS intercepts or notes during execution of an application program. Refer to Section 3.2.2 for a discussion of classes of exceptions. If the DBA specifies INTERCEPT, DBCS types an error message and forces

the application program to exit to the monitor – if an exception of the specified class occurs. Your program will continue to execute, however, if you type the CONTINUE monitor command. If the DBA specifies NOTE, DBCS types a message – again if an exception of the specified class occurs. DBCS does not stop execution of your application program.

DBCS can also keep you informed as to the exception conditions following the execution of any DML command. The exception condition code and other pertinent information is placed in special registers. The registers specific to each host language are described in Chapter 4 (for COBOL) and Chapter 5 (for FORTRAN). This section introduces the error special registers and describes the classes of exceptions defined in DBMS-20.[1] Refer to Appendix B for a list of exception codes and a detailed description of each code.

3.2.1 Error Special Registers

Five special registers are used to store exception-condition information. These registers are:

- Error Status
- Error Count
- Error Area
- Error Record
- Error Set

Each time a DML command is executed, the Error Status and Error Count registers are set to defined values. If no exception occurs during execution of a DML command, Error Status equals the null-string (a zero-word), and Error Count equals zero. The other three error special registers contain the values they had before execution of the DML command.

Should an exception occur during execution of a DML command, however, Error Status contains a code in the form

xxyy

where xx is a code identifying the statement or class of exception (see Section 3.2.2), and yy is a code identifying the exception. (See Appendix B.) Error Count equals 1. Error Area contains the name of the last area referenced. Except for the OPEN and CLOSE verbs – for which Error Record contains the null string – Error Record contains the following:

- Blanks if the exception occurs while DBCS is still processing the command argument list;
- Otherwise, the name of the object record type, except for

  1. a qualified DELETE
  2. a FIND NEXT RECORD of a specified set (an rse 3) in which the record name is not null, and
  3. a FIND OWNER.

  For these three cases Error Record contains the name of the last record operated upon.

Error Set is blank – unless at least one set operation has begun at the time the exception occurs. If a set operation has begun, Error Set contains the set name. Each of the following constitutes a set operation.

1. a FIND NEXT/PRIOR/OWNER of SET (rse 3 and rse 4)
2. each automatic insert during a STORE
3. each actual insert during an INSERT
4. each removal during a REMOVE
5. each automatic removal during a DELETE
6. each resorting of a set occurrence during a MODIFY.

[1] As noted in Chapter 1, the term exception is used in conformance with CODASYL terminology.

3.2.2 Classes of Exceptions

A number of classes of exceptions are defined for DBMS. These classes and their meanings are as follows:

BIND	exceptions that can occur during binding of the sub-schema; that is when DBCS enters SBIND, BIND, EBIND, and SETUSE. The BIND class of exceptions applies to the INVOKE DML command and to USE initialization. Refer also to Section 2.2.1 for a discussion of the INVOKE command. (Note that an INVOKE exception cannot be intercepted before the schema line is processed.)
CALL	exceptions that can occur when you explicitly call the journaling and SETDB and UNSET subprograms. Refer to Section 2.2.1 for a discussion of SETDB and UNSET and to Section 2.3 for a discussion of journaling.
HOST	exceptions that can occur with use of the IF predicates and the MOVE STATUS command.
SYSTEM	exceptions that can occur when a problem exists with the DBMS interface to the application program. Refer to exception codes 55 through 67 in Table B-2.
UPDATE	exceptions that can occur during execution of the updating verbs: DELETE, INSERT, MODIFY, REMOVE, and STORE.
UNANTICIPATED	all exceptions except 0307 and 0326.
ALL	any exceptions that can occur during execution of an application.

The DBA can use the BIND, CALL, SYSTEM, UPDATE, and ALL classes in his specification for the INTERCEPT and NOTE clauses.

Table 3-1 lists the DML statement-associated functions and codes.

Table 3-1 DML-Statement-Associated Functions and Codes

Statement	Code
HOST	00
CLOSE	01
DELETE	02
FIND	03
GET	05
INSERT	07
MODIFY	08
OPEN	09
REMOVE	11
STORE	12
BIND	15
CALL	16

3.3 DML STATEMENTS: DESCRIPTIONS AND FORMATS

This section lists the DML statements in alphabetical order and describes each statement. The descriptions include:

1. a delineation of function
2. an illustration of general format
3. pertinent technical notes
4. exception conditions and associated codes, and
5. an example of coding using the statement.

Each statement begins on a new page.

ACCESS

Function

Use the ACCESS statement to make available to one program-unit the data descriptions for a sub-schema invoked in another program-unit.

General Format

ACCESS SUB-SCHEMA sub-schema-name OF SCHEMA schema-name
[PRIVACY KEY FOR COMPILE IS key-1].

Technical Notes

1. The ACCESS statement can be used only once within each program-unit. That is, an ACCESS statement can, but need not, appear in any subprogram in the run-unit. It cannot appear in the same program-unit as an INVOKE statement.
2. Schema-name must be the name of a schema known to DBMS.
3. Sub-schema-name must refer to a sub-schema of the named schema.
4. You must specify a PRIVACY KEY FOR COMPILE if the sub-schema has a privacy lock declared for it. Refer to Section 1.5.4 for a discussion of privacy locks and keys.
5. Key-1 must conform to the data characteristics of privacy locks and keys. DBMS attempts to match the PRIVACY LOCK specified for the use of the sub-schema named with the PRIVACY KEY supplied by the program.
6. All records and data-items defined in the sub-schema are available in the UWA for reference in the FORTRAN program-unit containing the ACCESS statement. A COBOL program-unit must have the records and data-items that it will process passed to it from the calling program-unit. The call must explicitly include AREA-IDs, DIRECT keys and ALIASes, SYSCOM, and any other records or data-items you want to reference.
7. Some time earlier in the execution of the run-unit, an INVOKE statement must have been processed for the sub-schema being accessed. Refer to Sections 2.2.1 and 2.2.2 for a discussion of the use of INVOKE and ACCESS, respectively.

Example

```
ACCESS SUB-SCHEMA SUB1 OF SCHEMA BARNEX
PRIVACY KEY COMPILE SALEX.
```

CLOSE

Function

Use the CLOSE statement to relinquish control over the specified areas and make them available to other requesting run-units.

General Format

Format 1

```
CLOSE ALL.
```

Format 2

```
CLOSE AREA area-name-1 [area-name-2] ....
```

Format 3

```
CLOSE RUN-UNIT.
```

Format 4

```
CLOSE JOURNAL .
```

Technical Notes

1. The areas to be closed must be included in the sub-schema that was invoked by the run-unit.
2. Any area that was opened must be explicitly closed before the run is stopped.
3. All areas that are currently open for the run-unit may be closed at one time (Formats 1 and 3); individual areas may be closed as soon as processing within these areas is completed (Format 2).
4. Once an area has been closed, it may be reopened by the run-unit.
5. If you specify Format 3, all open areas will be closed and the journal file will be closed. The next run-unit opening the journal file will then append to it. (Refer to Section 2.3 on journaling.)
6. Specify Format 4 when you want to indicate to DBCS that all run-units have successfully concluded and the journal file is no longer necessary. DBCS will then overwrite the journal file the next time it is opened. If another run-unit has the journal file open at the time you execute Format 4, however, the effect will be that of Format 3.
7. If the journal file is not open, Formats 1, 3, and 4 of the CLOSE statement are equivalent.

Exception Conditions

1. If the CLOSE statement signals an exception, the system will not be backed-up to the state existing before the exception occurred.
2. After you have closed an area, you cannot reference that area nor any record occurrences within that area in a procedure. An attempt to make such reference returns a code indicating that the area is closed. (Refer to Appendix B for a list of exception codes.)

Example

```
CLOSE AREA MARKETING-AREA, PERSONNEL AREA.
```

DELETE

Function

Use the DELETE statement to make the object record occurrence unavailable for further processing by DML commands, and to remove the object record from all set occurrences in which it is a member. Note that using DELETE has additional potential effects; these are given in the technical notes below.

General Format

DELETE [record-name] [ONLY / SELECTIVE / ALL] .

Technical Notes

1. Record-name, if present, must refer to a record that has been defined in the sub-schema named in the program.
2. The object record occurrence of the DELETE statement is the current record of the run-unit.
3. The object record is removed from all set occurrences in which it is a member and it is then deleted; that is, made unavailable for further processing by a DML statement.
4. The unqualified form of DELETE deletes the object record only if it has no member records. Otherwise an exception condition is returned.
5. DELETE ONLY deletes the object record and all MANDATORY members of any sets for which it is the owner. It removes (see REMOVE) but does not delete its OPTIONAL members. If any of the deleted MANDATORY members are themselves the owners of any set occurrences, the DELETE statement treats such records as if each were the object record of a DELETE ONLY statement. Thus, all MANDATORY members of such sets are also deleted in turn causing this process to continue down the chain.
6. DELETE SELECTIVE has the same effect as DELETE ONLY except that:

   - OPTIONAL members may be deleted. They will be deleted if they do not participate as members in any other set occurrences.

   - All deleted records that are themselves the owners of any set occurrences are treated as if each were the object of a DELETE SELECTIVE statement.

7. DELETE ALL deletes the object record together with all of its member records, regardless of whether they are MANDATORY or OPTIONAL. The process continues down the hierarchy with all deleted records being treated as if each were the object of a DELETE ALL statement.
8. The current record of the run-unit becomes null. No other currency status information is altered. Thus, the object record and all other records deleted or removed remain as current of area-name, or record-name, and current of all set-names in which they were current prior to the execution of the DELETE statement.

Exception Conditions

1. When an exception occurs, the data base and User Working Area remain in the state existing before the attempted execution of the DELETE statement.
2. If there is no current record of the run-unit, Error Status returns code 0213.

3. If record-name is specified and the current record of the run-unit is not an occurrence of the named record type, Error Status returns code 0220. This is a debugging and documentation feature.
4. If the unqualified form of the DELETE imperative is attempted against the owner record of a non-empty set occurrence, Error Status returns code 0230.
5. If any of the record occurrences which would be deleted, removed, or modified as a result of the execution of the DELETE imperative are in an unopened area, Error Status returns code 0201.
6. If the object record or any record that would be deleted or removed as a result of the execution of a DELETE statement is located within an area that is open for RETRIEVAL, Error Status returns code 0209.
7. The sub-schema invoked must name

   a. All of the records that would be deleted or removed as a result of executing the DELETE statement;
   b. All of the sets from which any record is to be removed;
   c. All record occurrences referenced implicitly by the DELETE statement.

   Otherwise, Error Status returns code 0208.

8. If any record in the scope of a DELETE is current-of-run-unit of another run-unit, Error Status returns code 0240.

Example

```
DELETE CUSTOMER-RECORD ONLY.
```

END

Function

Use the END statement to end a FORTRAN program-unit containing DML statements.

General Format

END [progname].

Technical Notes

1. The END statement can be used only in FORTRAN programs.
2. Progname is a 1- to 6-character name for the program-unit.
3. If the progname is omitted, NONAME is assumed.
4. While use of the END statement is not required except to segregate multiple INVOKE and ACCESS statements, its use is recommended because it enables the FORTRAN preprocessor to print error summaries based on program units.

Example

```
END CORON.
```

FIND

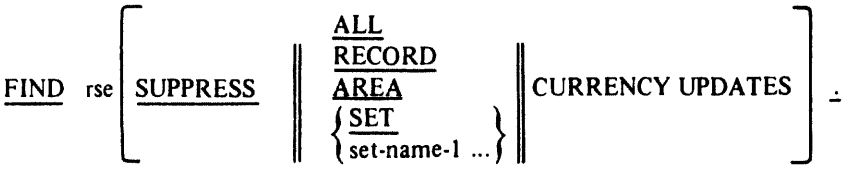
Function

Use the FIND statement

1. to establish the record occurrence specified by a record-selection-expression (rse) as the current record of the run-unit, and
2. to control whether or not it becomes

   a. current of the area in which it is stored,
   b. current of its record-type, and
   c. current of set for all set occurrences in which it participates as an owner or member.

General Format



Technical Notes

1. The five rse are described individually on the following pages.
2. All data-items, records, sets, and areas specified must be defined in the sub-schema named in the program.
3. Appropriate use of each FIND rse requires you to know the set relationships and record location modes defined within the sub-schema associated with a given program.
4. A successfully executed FIND statement results in the record occurrence selected becoming the current record of the run-unit. The contents of the record, however, are not available in the UWA until a GET statement is executed for the object record.
5. The SUPPRESS phrase is designed to specify the area, record, and set currency indicators that are to retain their existing values. The SUPPRESS ALL phrase prevents the update of all currency indicators except the current of run-unit.
6. If you do not use the optional SUPPRESS phrase, the object record also becomes the current record of its area, the current record of its record type, and the current record of all sets in which it is defined as an owner or in which it participates as a member.
7. For as long as a record is current-of-run-unit, it is retained by that run-unit. This retention applies, however, only if the area in which the record is found has been opened in UPDATE or RETRIEVAL usage-modes.

Exception Conditions

1. If exception conditions occur, the FIND statement is not successfully executed; the data base and the User Working Area remain in the state existing before the attempted execution, and the appropriate code is placed in the Error Status register.
2. If the object record is in an area that has not been opened, Error Status returns code 0301.
3. If there is no record of the type specified, Error Status returns code 0306.
4. If a database key is supplied or developed which is incompatible with the areas specified, Error Status returns code 0302.
5. If end-of-set or end-of-area condition is detected, Error Status returns code 0307.
6. If no record in the area satisfies the rse specified, Error Status returns code 0326.
7. If the referenced record, set, or area is not in the sub-schema, Error Status returns code 0308.

FIND rse 1

General Format

[record-name-1] USING identifier-1

Technical Notes

1. Identifier-1 must be defined USAGE DATABASE KEY, PICTURE 9(10) COMP (for COBOL), or INTEGER (for FORTRAN) and must be initialized with a database key value.
2. If record-name-1 is specified, the database key supplied must identify a record occurrence of record-name-1.

Exception Conditions

1. If the sought record is in an unopened area, Error Status returns code 0301.
2. If no record is identified by the database key supplied (i.e., the value of identifier-1), or if the database key identifies a record that is not an occurrence of record-name-1, Error Status returns code 0326.
3. If identifier-1 contains an invalid page number, Error Status returns code 0302.
4. If identifier-1 contains a value that DBCS could not have created (for example, it is on an uncreated page, or its line is equal to 0 or is greater than the maximum line number on the page), Error Status returns code 0356.

Example

```
FIND SALESMAN-RECORD USING SALKEY.
```

FIND rse 2

General Format

[OWNER IN set-name-2 OF] CURRENT OF { record-name-1 RECORD | set-name-3 SET | AREA-NAME-1 AREA | RUN-UNIT }

Technical Notes

1. Record-name-1 must be defined as a member of set-name-2. Set-name-2 and set-name-3 may be the same set-name or different set-names.
2. When the OWNER phrase is omitted, this rse selects the record occurrence that is the current record of the specified record, set, or area, or it selects the current record of the run-unit.
3. Use of the CURRENT OF RUN-UNIT form of the rse permits revision of currency status indicators that were previously suppressed.
4. When the OWNER phrase is present, the owner-record occurrence in set-name-2 is selected relative to current of record, set, area, or run-unit.

Exception Conditions

1. If the OWNER phrase is not used, and the currency indicator has been deleted or removed, Error Status returns codes 0317 or 0322 respectively.
2. If the OWNER phrase is used and the set occurrence no longer exists, Error Status returns code 0317.
3. If the specified current record does not currently participate as a member of any occurrence of set-name-2 and is not known by DBCS as the current record of set-name-2, Error Status returns codes 0317 or 0322.

FIND rse 3

General Format

```
{NEXT      }
{PRIOR     }
{FIRST     }  [record-name-3] RECORD OF {set-name-4 SET  }
{LAST      }                            {area-name-2 AREA}
{integer-1 }
{identifier-2}
```

Technical Notes

1. If record-name-3 and area-name-2 are stated, record-name-3 must be included in area-name-2. If both record-name-3 and set-name-4 are stated, record-name-3 must be defined as a member of set-name-4. Integer-1 must be a signed integer and cannot be zero. Identifier-2 must be an integer and cannot be zero.
2. If a set-name is specified, the set occurrence from which the object record is to be selected is identified by the current record of the specified set.
3. If the record-name-3 phrase is used, only occurrences of record-name-3 will be considered in evaluating the rse.
4. If the areas involved include occurrences of record types not known to the run-unit, only the records specified in the invoked sub-schema will be considered during evaluation of the rse.
5. When record-name-3 is used and NEXT/PRIOR RECORD OF AREA/SET is specified, the next or prior record is relative to the current of area/set. If you want to FIND NEXT/PRIOR of AREA/SET, then perform currency-setting activities on other records in the area/set, suppress area/set currency updates if you want to maintain your place in the same area/set and continue reading the records in that area/set. For example, assume that HDR and SUB are owner and member of SET1 and that HDR 0 is the current of area.

```
page 14     HDR 0          FIND NEXT HDR RECORD OF AREA1  AREA
                           GET HDR

page 15     HDR 1          FIND  NEXT  SUB  RECORD OF SET1 SET
            HDR 2          GET SUB
            SUB (H1)       will  cause  HDR  1  AND  SUB  (H1)
            SUB (H2)       records to be retrieved.  A  repeat
                           of  these  instructions  will cause
                           HDR 3
page 16     HDR 3          on  page  16 to be retrieved rather
            SUB (H3)       than HDR 2 on page 15.  To get  HDR
                           2,  suppress  area updates  in the
                           second FIND statement.
```

6. NEXT RECORD of area-name AREA means the record with the next higher database key relative to the current record of the named area.
7. PRIOR RECORD OF area-name AREA means the record with the next lower database key relative to the current record of the named area.
8. NEXT RECORD OF set-name SET means the subsequent record relative to the current record of the named set in the logical order of the set without regard to the database key sequence.

9. PRIOR RECORD OF set-name SET means the previous record relative to the current record of the named set in the logical order of the set regardless of the database key sequence.
10. FIRST RECORD OF area-name AREA is the record occurrence with the lowest database key in the named area.
11. LAST RECORD OF area-name AREA is the record occurrence with the highest database key in the named area.
12. FIRST RECORD OF set-name SET is the first member occurrence in terms of the logical order of the set. The record selected is the same as that selected if the current record of the set were the owner record and the NEXT RECORD of set-name SET rse were used.
13. LAST RECORD OF set-name SET is the last member record occurrence in terms of the logical order of the set. The record selected is the same as that selected if the current record of the set were the owner record and the PRIOR RECORD OF set-name SET rse were used.
14. Identifier-2 must be initialized with an integer prior to execution of the FIND statement.
15. Identifier-2 and integer-1 represent the ordinal count of the object record occurrence relative to the beginning, if positive, or ending, if negative, of a set occurrence or area. In other words, a negative value selects in the PRIOR direction and a positive value in the NEXT direction for the set occurrence or area.

Exception Conditions

1. Error Status returns code 0307 if there is no record with a higher database key (for NEXT of AREA rse), or if there is no record with a lower database key (for PRIOR of AREA rse). It also returns code 0307 if

   a. the current record is the first record in a set (for PRIOR rse)
   b. the current record is the last record in a set (for NEXT rse)
   c. the value of integer-1 or the contents of identifier-2 is greater than the number of record occurrences in the set occurrence or area specified.
   d. a FIND FIRST or LAST of AREA is specified for an empty area.

2. If a set occurrence is empty, or if integer-1 or identifier-2 is zero, Error Status returns code 0326.
3. If the referenced record, set, or area is not in the sub-schema Error Status returns code 0308.

Examples

```
FIND NEXT RECORD OF FIELD-SET SET.
```

FIND rse 4

General Format

OWNER RECORD OF set-name-5 SET

Technical Notes

1. The owner record of the current occurrence of set-name-5 is selected. It is exactly equivalent to using rse 2 and specifying the same set-name for set-name-2 and set-name-3.

Exception Conditions

1. If the current set occurrence no longer exists, Error Status returns code 0317.
2. If there is no current record of the type specified, Error Status returns code 0306.

Example

```
FIND OWNER RECORD OF CUSTOMER-SET SET;
SUPPRESS AREA UPDATES.
```

FIND rse 5

General Format

[NEXT DUPLICATE WITHIN] record-name-4 RECORD

Technical Notes

1. Use of this rse is restricted to record-types that have a CALC LOCATION MODE. Before issuing the FIND, move the desired key values to the appropriate key fields (object of LOCATION MODE clause) in the UWA. Verify the validity of this rse for a particular record-type with the DBA.

2. The NEXT DUPLICATE phrase is ignored if there is no current of run-unit, or the key or the record type of the current of run-unit does not match that of the record specified.

   When the NEXT DUPLICATE phrase is included and is meaningful, DBCS only considers records later in the came CALC list in its search for a record that has the UWA- specified key.

3. You must set the record AREA-ID to an appropriate value if the record may be in more than one area. This is true even if all of the areas are not open. The area in which you want to find the record must, however, be open.

Exception Conditions

1. Error Status returns code 0326 when DBCS cannot find a record that satisfies this rse.

2. Error Status returns code 0323 if the AREA-ID is applicable, but the given value is invalid.

Example

```
FIND PREDICTION-RECORD RECORD.
```

GET

Function

Use the GET statement to transfer the specified data-items of the object record occurrence into the User Working Area.

General Format

Format 1

```
GET [record-name].
```

Format 2

```
GET record-name data-name-1 [data-name-2] ....
```

Technical Notes

1. Record-name must refer to a record that has been defined in the sub-schema named in the INVOKE statement in the program.
2. Data-name-1, data-name-2, etc., must be the names of data-items defined as part of the named record. The record-name serves as an implicit major qualifier for the data-names.
3. The object of the GET imperative is the current record of the run-unit, as established by a previously-issued FIND (or STORE) statement. Therefore, the record-name given must be the current record of run-unit.
4. A GET statement must be executed before any reference can be made to the data of the object record in the User Working Area.
5. If you specify Format 1, all data-items defined in the named sub-schema for the object record are moved to the User Working Area. If you specify Format 2, only the specified data-items are moved to the User Working Area.

Exception Conditions

1. If any exceptions are encountered, the GET statement is not successfully executed. The data base and the UWA remain in the state existing before the attempted execution.
2. If there is no current record of the run-unit, Error Status returns code 0513.
3. If a record-name is specified and the current record of the run-unit is not an occurrence of the named record type, Error Status returns code 0520. This is a debugging feature.
4. If a specified data-name is not a field in the object record type, Error Status returns code 0504.

Example

```
GET SALESMAN-RECORD; SALESMAN, BASE-SALARY.
```

IF

Function

Use the IF statement to cause a condition to be evaluated. The subsequent action of the run-unit depends on whether the value of the condition is true or false.

General Format

Format 1

```
IF set-name-1 SET [NOT] EMPTY   { statement-1      }
                                { NEXT SENTENCE    }

[ ELSE   { statement-2      } ] .
         { NEXT SENTENCE    }
```

Format 2

```
IF RECORD [NOT] [ MEMBER ]   OF   { set-name-2 }   SET
                [ OWNER  ]        { ANY        }

{ statement-3      }          [ ELSE   { statement-4      } ] .
{ NEXT SENTENCE    }                   { NEXT SENTENCE    }
```

Technical Notes

1. The IF statement is part of the COBOL language, not the DML. Thus, it cannot be used in FORTRAN programs. Refer to Section 3.4 for the intrinsic functions that can be used in FORTRAN programs to perform the same operations as the IF statement.
2. Format 1 of the IF statement determines whether or not the object set occurrence has any members. The object set occurrence is determined by the current record of set-name-1. If you omit the NOT phrase, and the set occurrence does not have any member records, the condition is evaluated as true. If you omit the NOT phrase, and the set occurrence contains member records, the condition is evaluated as false. If you include the NOT phrase, the condition is reversed. Whether or not you use the NOT phrase, the condition is evaluated as true if there is no current record of set or the set-name is invalid.
3. Format 2 of the IF statement determines whether or not the current record of the run-unit participates as an owner or member, depending on whether you specify set-name-2 or ANY set. If you omit the NOT phrase, and specify the record as an owner or member, the condition is evaluated as true. If you omit the NOT phrase, and do not specify the record as an owner or member, the condition is evaluated as false. If you include the NOT phrase, the condition is reversed. Whether or not you use the NOT phrase, the condition is evaluated as false if there is no current record of the run-unit or if the set-name is invalid.
4. If you do not specify either the OWNER or MEMBER phrase in Format 2, the test is of the association as an owner or member of the object record with the specified set.
5. Rules pertaining to the execution of subsequent statements – depending on whether the condition is evaluated as true or false – are identical to those for other standard COBOL forms of the IF statement.

Exception Conditions

1. Only system exceptions are possible. Refer to Appendix B.

Example

```
IF RECORD OWNER OF ANY SET; GO TO TAG; ELSE NEXT SENTENCE.
```

INSERT

Function

Use the INSERT statement to make the object record a member of occurrences of the specified sets if the object record type is defined as an OPTIONAL AUTOMATIC, OPTIONAL MANUAL, or MANDATORY MANUAL member of those sets.

General Format

Format 1

INSERT [record-name] INTO set-name-1 [set-name-2] ....

Format 2

INSERT [record-name] INTO ALL SETS.

Technical Notes

1. Record-name and all set-names must be defined in the sub-schema named in the program.
2. In Format 1, the object record must be defined as an OPTIONAL AUTOMATIC, OPTIONAL MANUAL, or MANDATORY MANUAL member of the specified sets.
3. In Format 2, the object record must be defined as an OPTIONAL AUTOMATIC, OPTIONAL MANUAL, or MANDATORY MANUAL member of at least one set in the invoked sub-schema.
4. The object record occurrence of the INSERT statement is the current record of the run-unit.
5. If you use Format 1, the object record is inserted into the object set occurrence of each set-name specified in accordance with the set-ordering criteria defined in the schema. For each set named, the object set occurrence is determined by the current of set.
6. If you use Format 2, the object record is inserted into the appropriate occurrence of each set included in the invoked sub-schema; the object record must be defined as an OPTIONAL AUTOMATIC, OPTIONAL MANUAL, or MANDATORY MANUAL member, and must not already be a member of the set. The specific occurrence of each set is determined by the current record of set for each of the set-names involved. The object record is inserted into each set occurrence in accordance with the set-ordering criteria specified in the schema.

Exception Conditions

1. When an exception occurs, the data base and the User Working Area remain in the state existing before the attempted execution of the INSERT statement, and the appropriate exception-condition code is made available.
2. If there is no current record of the run-unit, Error Status returns code 0713.
3. For Format 1, if the object record is not defined as an OPTIONAL AUTOMATIC, OPTIONAL MANUAL, or MANDATORY MANUAL member of each set in the schema, Error Status returns code 0714.
4. If the object record, when inserted, would violate a DUPLICATES NOT ALLOWED clause for any record or set involved, Error Status returns code 0705.
5. If there is no current record of the specified set type, Error Status returns code 0706.
6. If the object record is already a member of any occurrence of a set explicitly named in Format 1, or if it is already a member of an occurrence of each set implicitly specified by the use of Format 2, Error Status returns code 0716. (For Format 2, this includes an object record not defined as an OPTIONAL AUTOMATIC, OPTIONAL MANUAL or MANDATORY MANUAL member of any set in the sub-schema.)

7. If any object set occurrence has been deleted, Error Status returns code 0717.
8. If the record-name is specified and the current record of the run-unit is not an occurrence of the named record type, Error Status returns code 0720. This is a debugging feature.
9. If the object record or any record occurrence affected by the INSERT statement is located in an area that is open for RETRIEVAL, Error Status returns code 0709.
10. If the object record is not an occurrence of the member record type of a specified set, Error Status returns code 0722.
11. If both temporary and permanent areas are referenced, Error Status returns code 0724.

Example

```
* INSERT SALESFIELD-RECORD INTO ALL SETS.
```

INVOKE

Function

Use the INVOKE statement to specify the sub-schema that provides the description of that portion of the data base known to the program. Using INVOKE also assigns appropriate UWA locations during compilation and binds them for each execution of the program.

General Format

```
INVOKE SUB-SCHEMA sub-schema-name OF
       SCHEMA schema-name
       [PRIVACY KEY FOR COMPILE IS key-1].
```

Technical Notes

1. The INVOKE statement can be used at most once in each program-unit within a run-unit. That is, an INVOKE statement can, but need not, appear in the main program and any subprograms in the run-unit. Refer to Chapter 4 for the description of the placement and use of the INVOKE statement in COBOL programs; refer to Chapter 5 for FORTRAN programs.
2. Schema-name must be the name of a schema known to DBMS.
3. Sub-schema-name must refer to a sub-schema for the named schema, and must be part of that data base.
4. Sub-schema names must be unique within any run-unit that invokes more than one sub-schema.
5. PRIVACY KEY FOR COMPILE is required if the sub-schema has a privacy lock declared for it. Privacy locks and keys are discussed in Section 1.5.4.
6. Key-1 must conform to the data characteristics of privacy locks and keys. DBMS attempts to match the PRIVACY LOCK specified for the use of the sub-schema named with the PRIVACY KEY supplied by the program.
7. Once the INVOKE statement is executed, all UWA locations that are necessary for data manipulation have been communicated to DBCS. This means that all records and data-items defined in the sub-schema are available in the UWA for appropriate reference by both DML and standard host-language commands.

Exception Conditions

1. If the compile-time and run-time versions of the schema file differ, Error Status returns code 1500.
2. Error Status can return codes 1531 through 1536 only during binding. Refer to Appendix B.

Example

```
INVOKE SUB-SCHEMA SUB1 OF SCHEMA BARHEX
     PRIVACY KEY COMPILE SALEX.
```

MODIFY

Function

Use the MODIFY statement to replace the value of all or specified data-items of the object-record occurrence in the data base with values from the User Working Area (UWA). MODIFY also alters intraset position so as to maintain the data base in accordance with the set-ordering criteria specified in the schema.

General Format

Format 1

MODIFY [record-name].

Format 2

MODIFY record-name data-name-1 [data-name-2] ....

Technical Notes

1. The named record and all data-items identified by their data-names must be defined in the sub-schema that was invoked.
2. In Format 2, record-name serves as the implicit major qualifier for data-name-1, data-name-2, etc.
3. The object record occurrence of the MODIFY statement is the current record of the run-unit. If you use Format 1, all data-items in the object record are modified with values from the UWA.
4. If you use Format 2, only the data-items specified are modified from the UWA. All other data-items in the object record occurrence in the data base remain unchanged.
5. You must initialize all data-items involved in the UWA with the required values prior to execution of the MODIFY statement.
6. If any of the modified data-items in the object record are defined as sort-control items for any set occurrences in which the object record is a member, modification causes the intraset occurrence position of the object record to be examined; if necessary, the object record is removed and re-inserted in the set occurrences to maintain the set order specified in the schema. The current occurrence of the set-name involved remains as the current occurrence. If the current of run-unit is the current of the set-name involved, it remains as the current of set-name. If not, it becomes the current of that set-name.
7. If any modified data-items are CALC keys of the object record, the record is relinked to the appropriate CALC chain.

Exception Conditions

1. When an exception occurs, the data base and the User Working Area remain in the state existing before the attempted execution of the MODIFY statement, and the appropriate code is made available in the Error Status register.
2. If there is no current record of the run-unit, the MODIFY statement is not executed and Error Status returns code 0813.
3. If record-name is specified and the current record of the run-unit is not an occurrence of the named record-type, Error Status returns code 0820. This is a debugging feature.
4. If the insertion of the object record under any of the conditions described above would violate the condition that any of the sets or records involved are not allowed duplicate occurrences, the MODIFY statement is not executed and Error Status returns code 0805.

5. If the object record, or any record occurrence affected by the execution of the MODIFY statement, is located in an area that is open for RETRIEVAL, Error Status returns code 0809.
6. If both temporary and permanent areas are referenced, Error Status returns code 0824.
7. If the data-name used is invalid or inconsistent, Error Status returns code 0804.
8. If the referenced record, area, or set is not in the invoked sub-schema, Error Status returns code 0808.

Example

```
MODIFY CUSTOMER-RECORD;  CREDIT-STATUS.
```

MOVE STATUS

Function

Use the MOVE STATUS statement to save the contents of the specified currency status indicators.

General Format

MOVE CURRENCY STATUS FOR { RUN-UNIT / record-name RECORD / area-name AREA / set-name SET } TO identifier-1.

Technical Notes

1. The record-name, area-name, or set-name must be included in the sub-schema named in the program.
2. Identifier-1 must refer to a data item defined as PICTURE 9(10) COMP or DATABASE-KEY (for COBOL) or INTEGER (for FORTRAN).
3. If you use the RUN-UNIT phrase, place the database key for the current record of the run-unit in identifier-1. The current record of the run-unit is not altered. If you specify record-name, area-name, or set-name, place the database key for the current record of record-name, area-name, or set-name in identifier-1. The current record of record-name, area-name, or set-name is not altered.
4. MOVE STATUS alters the special registers Area-Name and Record-Name to describe the currency indicator being moved.

Example

```
MOVE CURRENCY STATUS FOR RUN-UNIT TO STAKEY.
```

OPEN

Function

Use the OPEN statement to make one or more areas available for processing and to specify the usage-mode for these areas.

General Format

Format 1

```
OPEN ALL [ USAGE-MODE is { RETRIEVAL                  } ].
                         { UPDATE                     }
                         { EXCLUSIVE UPDATE           }
                         { EXCLUSIVE RETRIEVAL        }
                         { PROTECTED UPDATE           }
                         { PROTECTED RETRIEVAL        }
```

Format 2

```
OPEN AREA area-name-1 [area-name-2] ...

    [USAGE-MODE is (       )]

    [PRIVACY KEY is key-1].
```

Format 3

```
OPEN JOURNAL USAGE-MODE [EXCLUSIVE] UPDATE.
```

Technical Notes

1. Each area named in an OPEN statement must be defined in the sub-schema that was invoked.
2. OPEN ALL means that every area referenced in the sub-schema will be opened in the USAGE-MODE specified. It also means that a failure to open any one area means that no area is opened. This format implies that there are no privacy locks on the areas.
3. PRIVACY KEY is required if the area has a privacy lock declared for it. Privacy locks and keys are discussed in Section 1.5.4.
4. Key-1 must conform to the data characteristics of privacy locks and keys. DBMS attempts to match the PRIVACY LOCK specified for the area named with the PRIVACY LOCK supplied by the program.
5. The USAGE-MODE clause in Format 2 is the same as that in Format 1.
6. USAGE-MODE is RETRIEVAL allows concurrent run-units to open the same area with any usage-mode other than EXCLUSIVE UPDATE/RETRIEVAL. This run-unit will not be able to STORE, MODIFY, DELETE, INSERT, or REMOVE.
7. USAGE-MODE is UPDATE allows concurrent run-units to open the same area with any usage-mode other than EXCLUSIVE UPDATE/RETRIEVAL or PROTECTED UPDATE/RETRIEVAL. This run-unit will be able to STORE, DELETE, MODIFY, INSERT, or REMOVE.
8. USAGE-MODE is EXCLUSIVE RETRIEVAL prevents concurrent run-units from interacting with the same area in any usage-mode. This run-unit will not be able to STORE, DELETE, MODIFY, INSERT, or REMOVE.
9. USAGE-MODE is EXCLUSIVE UPDATE prevents concurrent run-units from interacting with the same area in any usage-mode. This run-unit will be able to STORE, DELETE, MODIFY, INSERT, or REMOVE.

10. USAGE-MODE is PROTECTED RETRIEVAL prevents concurrent run-units from update and allows concurrent retrieval. This run-unit cannot STORE, DELETE, MODIFY, INSERT or REMOVE.
11. USAGE-MODE is PROTECTED UPDATE prevents concurrent run-units from update and allows concurrent retrieval. This run-unit can STORE, DELETE, MODIFY, INSERT, or REMOVE.
12. RETRIEVAL is assumed if no usage-mode is specified.
13. Use Format 3 to communicate to DBCS if the journal is to be shared when you are opening an area for update. Include the word EXCLUSIVE only if you will not be sharing the journal file; that is, you will not be backing up any areas opened in UPDATE usage-mode.
14. If used, Format 3 must appear before any areas opened for update. If Format 3 has not been used, the following rules apply when the first OPEN is attempted in an update usage-mode. If the area is opened in the EXCLUSIVE or PROTECTED UPDATE usage-modes, DBCS simulates an OPEN JOURNAL USAGE-MODE EXCLUSIVE UPDATE format. If the area is opened in the UPDATE usage-mode, DBCS simulates the OPEN JOURNAL USAGE-MODE UPDATE format.
15. All usage-modes specified in the OPEN imperative remain in effect until the execution of a CLOSE statement for the opened areas.
16. An area must be opened to enable processing of data in that area by any of the other DML statements discussed in this chapter.

Exception Conditions

1. An attempt to open an area in UPDATE usage-mode after opening the journal in EXCLUSIVE UPDATE causes Error-Status to return code 0938.
2. Any attempt to execute an OPEN statement that would result in a usage-mode conflict for an area causes Error-Status to return code 0940. Table 3-2 indicates usage-mode conflicts by an 'N' and permitted concurrency of run-units by a 'Y'.
3. If a privacy breach occurs, Error Status returns code 0910.
4. If a run-unit attempts to open an area which is already open, Error Status returns code of 0928.

Table 3-2 Usage-Mode Conflicts for OPEN

	CURRENT STATE OF AREA						
Requested Usage-Mode	**Not Opened**	**RETRIEVAL**	**UPDATE**	**PROTECTED RETRIEVAL**	**PROTECTED UPDATE**	**EXCLUSIVE RETRIEVAL**	**EXCLUSIVE UPDATE**
RETRIEVAL	Y	Y	Y	Y	Y	N	N
UPDATE	Y	Y	Y	N	N	N	N
PROTECTED RETRIEVAL	Y	Y	N	Y	N	N	N
PROTECTED UPDATE	Y	Y	N	N	N	N	N
EXCLUSIVE RETRIEVAL	Y	N	N	N	N	N	N
EXCLUSIVE UPDATE	Y	N	N	N	N	N	N

Example

```
OPEN AREA MARKETING-AREA, PERSONNEL-AREA;
     USAGE-MODE IS EXCLUSIVE UPDATE.
```

REMOVE

Function

Use the REMOVE statement to cancel the membership of the object record in the occurrences of the specified set-names in which it currently participates as a member. The object record must be defined (in the schema) as an OPTIONAL member of the sets named.

General Format

Format 1

REMOVE [record-name] FROM set-name-1 [set-name-2] ....

Format 2

REMOVE [record-name] FROM ALL SETS.

Technical Notes

1. Record-name and all set-names must be defined in the invoked sub-schema.
2. The object record of the REMOVE statement is the current record of the run-unit.
3. If you use Format 1, the object record must be defined (in the schema) as an OPTIONAL member of the specified sets.
4. If you use Format 2, the object record must be defined (in the schema) as an OPTIONAL member of at least one set included in the invoked sub-schema.
5. The object record must also participate as a member in an occurrence of at least one of the object sets.
6. No change occurs to any currency information maintained by the DBCS.

Exception Conditions

1. When an exception occurs, the data base and the User Working Area remain in the state existing before the attempted execution of the REMOVE statement, and Error Status returns the appropriate code.
2. If the current record of the run-unit is not known, Error Status returns code 1113.
3. If record-name is specified and the current record of the run-unit is not an occurrence of the named record type, Error Status returns code 1120. This is a debugging feature.
4. If the object record is not defined as an optional member of all of the specified set-names in Format 1, Error Status returns code 1115.
5. If no sets are actually specified by use of Format 2, Error Status returns code 1122.
6. If the object record does not participate as a member in an occurrence of at least one of the sets specified in Format 1, or implied by use of Format 2, Error Status returns code 1122.
7. If the object record, or any record occurrence affected by the REMOVE statement is located in an area that is open for RETRIEVAL, Error Status returns code 1109.

Example

```
REMOVE SALESFIELD-RECORD FROM FIELD-SET.
```

STORE

Function

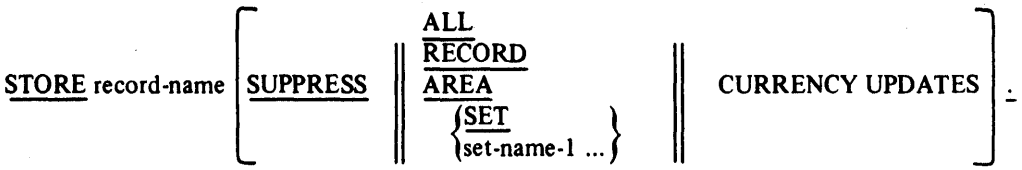
Use the STORE statement to

1. acquire space and a database key for a new record occurrence in the data base
2. cause the values of the appropriate data-items in the User Working Area to be included in the occurrence of the object record in the data base
3. insert the object record into all sets in the sub-schema for which it is defined as an AUTOMATIC member
4. establish a new set occurrence for each set for which the object record is defined as owner in the schema.

STORE also establishes the object record as the current record of the run-unit, and controls whether or not it becomes

1. the current record of the area in which it is stored,
2. the current record of its record-type, and
3. the current record of set for all set-types in which it is specified as an owner or AUTOMATIC member.

General Format



Technical Notes

1. The invoked sub-schema must include the named record, all sets in which the named record is defined as an AUTOMATIC member, and all data items, records, and sets necessary for correct placement of the named record in the data base.
2. A database key and space for the object record are allocated on the basis of the description of the record in the sub-schema invoked by the run-unit and the values you supply.
3. You must ensure that the following data-items in the User Working Area are initialized with the appropriate values:

   a. All data-items included in the object record of the STORE statement;
   b. Any control data-items and Area IDs in all owner records of the set types in which this record is defined as an AUTOMATIC member that have set occurrence selection LOCATION MODE OF OWNER. Verify with your Data Administrator which are the control data-items.

4. The object record occurrence is inserted into a set occurrence for each set type in which the record type is defined as an AUTOMATIC member. The ordering rules for the set govern the insertion point of the object record in all of the relevant set occurrences.
5. The object record is established as the owner of a set occurrence for each set type in which it has been defined as an owner. These set occurrences are empty at this time; that is, they have no member records.

6. The successfully stored record occurrence becomes the current record of the run-unit.
7. If you do not use the SUPPRESS phrase, the object record also becomes the current record of the area in which it is placed; the current record of its record-type; and the current record of all set-types in which it is defined as an owner or AUTOMATIC member.
8. The SUPPRESS phrase provides the facility to selectively prevent the object record from becoming the current record of the area, of its record type, and of any or all of the sets in which it is defined as either an owner or an AUTOMATIC member. SUPPRESS ALL prevents the update of all currency indicators. However, use of the SUPPRESS phrase does not prevent the object record from becoming the current record of the run-unit.

Exception Conditions

1. If any of the following conditions are encountered, the STORE statement is not successfully executed, the data base and UWA remain in the state existing prior to the attempted execution, and Error Status returns the appropriate exception code.
2. If the object record is to be stored in an area that is not open, Error Status returns code 1201.
3. If a database key is inconsistent with the relevant area, Error Status returns code 1202.
4. If the record to be stored would violate a DUPLICATES NOT ALLOWED clause defined for any of the records or sets involved, Error Status returns code 1205.
5. If the set type owned by the object record type is not in the invoked sub-schema, Error Status returns code 1208.
6. If the object record, or any record occurrence affected by the STORE statement, is located in an area that is open for RETRIEVAL, Error Status returns code 1209.
7. If a database key or data space is not available, Error Status returns code 1211.
8. If an illegal area name is passed in an AREA-ID, Error Status returns code 1223.
9. If both temporary and permanent areas are referenced, Error Status returns code 1224.
10. If a set occurrence that meets the set selection criteria (for any of the set-names involved) cannot be found, Error Status returns code 1225.

Example

```
STORE SALESMAN-RECORD SUPPRESS AREA UPDATES.
```

USE FOR COBOL

Function

Use the USE statement to specify procedures to be executed when specified exceptions occur.

General Format

USE IF ERROR-STATUS [IS integer-1 [integer-2] ...]

Technical Notes

1. Integer-1, integer-2... may be valid, four-digit exception codes.
2. The codes in each USE statement must be unique; that is, different USE statements cannot specify the same integers. If no integer is specified, it must be the last USE condition specified and will apply to all exception codes not specified in preceding USE statements.
3. If the execution of a DML command results in a specified exception, the procedure following the USE statement is executed.
4. To cause the procedure following the USE statement to be executed for every exception other than zero, omit specific exception codes from the USE statement.
5. To cause the procedure following the USE statement to be executed for all classes of exceptions pertaining to a specified DML command, use the statement code followed by 00 (for example, 0300 for all exceptions pertaining to FIND).
6. To cause the procedure following the USE statement to be executed for all DML statements pertaining to a specified exception code, use the exception code preceded by 00 (for example, 0062).
7. If an EXIT or an END DECLARATIVE statement is encountered in the execution of a USE procedure, control returns to the statement immediately following the DML command that resulted in the exception.
8. If the execution of a DML command results in an exception not specified by a USE statement, control proceeds to the statement following that DML command unless otherwise specified in this document. You determine occurrence of the exception by testing the special register ERROR-STATUS. Refer also to Section 3.2 for a discussion of the INTERCEPT and NOTE phrases.
9. USE statements should be in the main program or in a subroutine called only once.

Exception Conditions

1. The BIND statement code applies to USE initialization. Refer to Section 3.2.2 for a discussion of the classes of exceptions.

Example

```
USE IF ERROR-STATUS IS 0301
```

USE FOR FORTRAN

Function

Use the USE statement to specify the procedure to be executed when specified exceptions occur.

General Format

```
USE subprogram-name IF {ERROR-STATUS} [IS integer-1 [integer-2]...].
                       {ERSTAT      }
```

Technical Notes

1. Integer-1, integer-2... may be valid, four-digit exception codes.
2. Subprogram-name must refer to a FORTRAN subprogram that is part of the run-unit.
3. Different USE statements must specify different integers. If no integer is specified, it must be the last condition specified; it will apply to all exception codes that have not been in a previously executed USE statement.
4. If the execution of a DML imperative results in a specified exception, the specified subprogram is executed.
5. To cause the subprogram to be executed for every exception other than zero, omit specific exception codes from the USE statement.
6. To cause the procedure following the USE statement to be executed for all classes of exceptions pertaining to a specified DML command, use the statement code followed by 00 (for example, use 0300 for all exceptions pertaining to FIND).
7. To cause the procedure following the USE statement to be executed for all DML statements pertaining to a specified exception code, use the exception code preceded by 00 (for example, 0062).
8. When a RETURN statement is executed in the subprogram specified in the USE statement, control returns to the statement immediately following the DML imperative that resulted in the exception.
9. If the execution of a DML imperative results in an exception not specified by a USE statement, control proceeds to the statement following that DML imperative unless otherwise indicated in this document. The occurrence of the exception is determined by testing the special register ERSTAT.
10. The USE statement for FORTRAN is part of the DML and not part of the FORTRAN language.
11. Use statements should be in the main program or in a subroutine called only once. They should immediately precede the first procedural statement in the program.
12. A total of 16 USE conditions can be applied to one sub-schema.

Example

```
USE ERR IF ERSTAT IS 0319.
```

Exception Conditions

1. The BIND statement code applies to USE initialization. Refer to Section 3.2.2 for a discussion of the classes of exceptions.
2. If more than 16 USE conditions are specified, Error Status returns code 1535.

3.4 FORTRAN INTRINSIC FUNCTIONS

Four FORTRAN intrinsic functions have been provided for DBMS usage. Use them to determine whether or not the current record of the run-unit is the owner of a specified set, a member of a specified set, a tenant (i.e., either the owner or a member) of a specified set, or if the specified set is empty (i.e., contains no member records). Each function is described separately on the following pages.

EMPTY Function

Function

Use the EMPTY function to determine whether or not an occurrence of the specified set has any member-record occurrences.

General Format

EMPTY (set-name-1)

Technical Notes

1. If the current occurrence of the specified set contains no member-record occurrences, a value of .TRUE. is returned. If the current occurrence of the specified set contains at least one member-record occurrence, a value of .FALSE. is returned.
2. The function is evaluated as .TRUE. if there is no current record of the set type or if the set-name is invalid.

Example

```
IF (EMPTY('CUST-SET')) GO TO 10
```

MEMBER Function

Function

Use the MEMBER function to determine whether or not the current record of the run-unit is a member of an occurrence of the specified set or of any set.

General Format

MEMBER (set-name-1)

Technical Notes

1. Set-name-1 must be either a string or 0.
2. If you specify 0 as set-name-1, any set will satisfy the check for membership.
3. If the current record of the run-unit is a member of an occurrence of set-name-1 (or of any set if 0 is specified), a value of .TRUE. is returned. If the current record of the run-unit is not a member of an occurrence of set-name-1 (or of any set if 0 is specified), a value of .FALSE. is returned.
4. If set-name-1 is invalid or if there is no current record of the run-unit, a value of .FALSE. is returned.

Example

```
IF (MEMBER('DEPT-EMPLOY') .AND. EMPNAM .EQ. NAME) GO TO 999
```

OWNER Function

Function

Use the OWNER function to determine whether or not the current record of the run-unit is an owner of an occurrence of the specified set or of any set.

General Format

OWNER (set-name-1)

Technical Notes

1. Set-name-1 must be either a string or 0.
2. If you specify 0 as set-name-1, any set will satisfy the check for ownership.
3. If the current record of the run-unit is the owner of an occurrence of set-name-1 (or of any set if 0 is specified), a value of .TRUE. is returned. If the current record of the run-unit is not the owner of an occurrence of set-name-1 (or of any set if 0 is specified), a value of .FALSE. is returned.
4. If set-name-1 is invalid or if there is no current record of the run-unit, a value of .FALSE. is returned.

Example

```
IF(OWNER(0) .OR. MEMBER('CST')) GO TO 800
```

TENANT Function

Function

Use the TENANT function to determine whether or not the current record of the run-unit is the owner or a member of an occurrence of the specified set or of any set.

General Format

TENANT (set-name-1)

Technical Notes

1. Set-name-1 must be either a string or 0.
2. If you specify 0 as set-name-1, any set will satisfy the check for tenancy.
3. If the current record of the run-unit is the owner or a member of an occurrence of set-name-1 (or of any set if 0 is specified), a value of .TRUE. is returned. If the current record of the run-unit is not the owner or a member of an occurrence of set-name-1 (or of any set if 0 is specified), a value of .FALSE. is returned.
4. If set-name-1 is invalid or if there is no current record of the run-unit, a value of .FALSE. is returned.

Example

```
IF(TENANT('OVEP')) 300, 400
```

CHAPTER 4

USING THE DML IN COBOL PROGRAMS

You can embed DBMS Data Manipulation Language statements in a COBOL program (among COBOL statements). You must be careful, however, to follow the conventions discussed in Section 4.2 for placing the DML statements. After writing the program, you can then compile, load, and execute it just as you would any other COBOL program. The COBOL compiler performs the necessary translation of the DML statements along with the other COBOL statements.

4.1 BUILDING A COBOL-DML PROGRAM

Creating an executable application program using the DML with COBOL involves a series of steps. These are illustrated in Figure 4-1; they involve

1. Creating a new program (or using an existing program)
2. Compiling the program
3. Loading the relocatable object code
4. Optionally saving the loaded object code
5. Executing the run-unit.

4.2 PLACING DML STATEMENTS WITHIN COBOL

As already noted, you must be sure to place the DML statements in your COBOL program in accordance with prescribed rules. (Refer also to the *DECsystem-10 COBOL Programmer's Reference Manual*, Section 5.) Begin your COBOL program with the standard COBOL IDENTIFICATION and ENVIRONMENT DIVISIONs. These remain unchanged from those used in standard COBOL. Then proceed to the DATA DIVISION discussed in Section 4.2.1 under the INVOKE statement.

4.2.1 The INVOKE Statement

The COBOL DATA DIVISION incorporates the link to the data base. This link is called the SCHEMA SECTION. Place the INVOKE statement in the SCHEMA SECTION in the DATA DIVISION of a COBOL program-unit. The SCHEMA SECTION must follow the FILE SECTION, if the latter is present, and must precede all other sections in the DATA DIVISION.

When the INVOKE statement is compiled, the compiler creates a User Working Area (UWA) and, in effect, places it in the WORKING-STORAGE SECTION. The UWA contains record descriptions copied from the invoked subschema of the data base. In addition to the record descriptions, the UWA also contains a group of special status registers. These registers have the following descriptions.

```
01 SYSCOM.
02      AREA-NAME,          PIC X(30) USAGE DISPLAY-7.
02      RECORD-NAME,        PIC X(30) USAGE DISPLAY-7.
02      ERROR-STATUS,       PIC 9(5) USAGE DISPLAY-7.
02      ERROR-SET,          PIC X(30) USAGE DISPLAY-7.
02      ERROR-RECORD,       PIC X(30) USAGE DISPLAY-7.
02      ERROR-AREA,         PIC X(30) USAGE DISPLAY-7.
02      ERROR-COUNT         PIC 99, USAGE COMP.
```

These registers are used to store the names of the last area and record affected by a MOVE statement; the error status; the names of the set, record, and area where an exception occurs; and a 1 if an exception occurs. Refer also to Section 3.2 for a discussion of exception handling in DBMS.

When an exception occurs during execution of a DML statement, ERROR-STATUS contains a 4-digit code that describes the exception. These codes and their meanings are discussed in Appendix B. ERROR-AREA, ERROR-SET, and ERROR-RECORD contain the names of the area, set, and record in which the exception occurred if they are

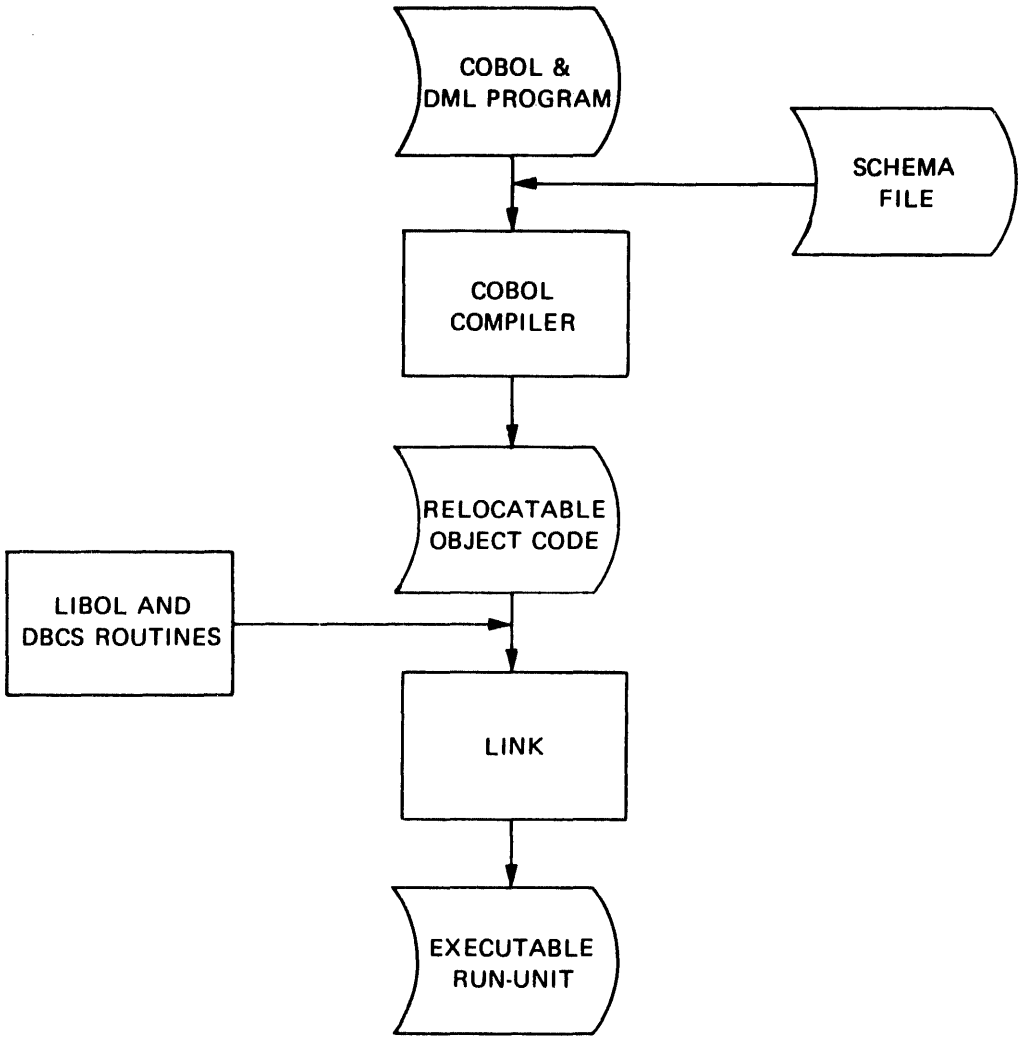


Figure 4-1 Program-Building Process for COBOL with DML

known to DBCS. ERROR-COUNT contains 1 if an exception occurs during execution of a DML statement; otherwise it contains 0. AREA-NAME and RECORD-NAME are used to store the names of the area and record last processed by a MOVE STATUS statement.

For further information about the INVOKE statement, refer to Section 2.2.1 and to Chapter 3.

4.2.2 The ACCESS Statement

Place the ACCESS statement in a subprogram when you want that subprogram to access the data in a sub-schema previously invoked in another program-unit. When the subprogram is called from a program-unit containing an INVOKE statement (or another ACCESS statement), the calling program-unit can pass as arguments to the called subprogram the special registers and the names of the records that the subprogram will be referencing. The ACCESS statement, in effect, causes an image of the UWA from the calling program-unit to be placed in the LINKAGE SECTION of the called subprogram. Note that use of a sub-schema from another program-unit does not violate the rule that subprograms cannot perform I/O on files in another program-unit. This is because areas are not files in the COBOL sense.

You must place the ACCESS statement, like the INVOKE statement, in the SCHEMA SECTION in the subprogram. The SCHEMA SECTION must follow the FILE SECTION, if the latter is present, and precede all other sections in the DATA DIVISION. Only one ACCESS statement can appear in a subprogram, and it cannot appear in a subprogram with an INVOKE statement. These restrictions are applied because only one sub-schema can be referenced in a given program-unit.

4.2.3 Other DML Statements

The other DML statements, CLOSE, DELETE, FIND, GET, IF, INSERT, MODIFY, MOVE STATUS, OPEN, REMOVE, STORE, and USE must appear in the PROCEDURE DIVISION of the COBOL program. You can place these DML statements anywhere in the PROCEDURE DIVISION. The USE statement is an exception. The USE statement must appear in the DECLARATIVES SECTION in the PROCEDURE DIVISION. Place the OPEN statement such that it is the first DML statement executed in the PROCEDURE DIVISION. The other DML statements cannot be successfully executed until an area is opened.

4.3 EXAMPLES

The examples shown on the following pages illustrate use of the DML statements in COBOL programs. Each example is preceded by a description of the example, and followed by numbered explanatory notes. These notes refer to statements or sections of the program that are denoted by a marker in the form:

_(1)

All programs are concerned with the same sub-schema of a schema called BARHEX. The name of the sub-schema is SUB-SCHEMA-1, and its privacy key is SALEX. Included in the sub-schema are two areas, six record types, and five sets. Figure 4-2 shows the schema BARHEX and the sub-schema SUB-SCHEMA-1. Note again that the Data Base Administrator, not the application programmer, is expected to establish and maintain the DDL definitions of the data base records, areas, sets, and sub-schemas. Refer to the schematic notation for the BARHEX schema in Figure 1-5 to assist you in visualizing retrieval of specific record types and movement within sets for the program examples that follow.

The sample programs in this section show how to walk through the structured data in the data base to find a desired record occurrence; how to retrieve, modify, and store data; and how to use data from the data base to write a report.

```
RECORDS-PER-PAGE 36.

ASSIGN PERSONNEL-AREA TO FILE1
FIRST PAGE IS 1
LAST PAGE IS 21
PAGE SIZE IS 256 WORDS.

ASSIGN MARKETING-AREA TO FILE2
FIRST PAGE IS 101
LAST PAGE IS 201
PAGE SIZE IS 256 WORDS.

SCHEMA NAME IS BARHEX.

AREA NAME IS PERSONNEL-AREA.
AREA NAME IS MARKETING-AREA.
```

Figure 4-2 The DML with COBOL: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

```
RECORD NAME IS SALESMAN-RECORD
          LOCATION MODE IS DIRECT IDENT1
          WITHIN PERSONNEL-AREA.

02        SALESMAN          PIC X(30).
02        HOME-ADDRESS      PIC X(25).
02        HOME-CITY         PIC X(15).
02        HOME-STATE        PIC XX.
02        HOME-ZIP          PIC 9(5).
02        HOME-PHONE        PIC X(12).
02        SS-NUMBER         PIC X(11).
02        BASE-SALARY       PIC 9(5)V99.
02        HIRING-DATE       PIC X(8).

RECORD NAME IS QTR-COMMISSION-RECORD
          LOCATION MODE IS VIA COMMISSION-SET
          WITHIN PERSONNEL-AREA.

02        QTR               PIC X(6).
02        COMMISSION        PIC 9(5)V99.
02        BONUS             PIC 9(5)V99.

RECORD NAME IS SALESFIELD-RECORD
          LOCATION MODE IS DIRECT IDENT2
          WITHIN MARKETING-AREA.

02        FIELD-NUMBER      PIC 9999.
02        FIELD-LOCALE      PIC X(30).

RECORD NAME IS CUSTOMER-RECORD
          LOCATION MODE IS CALC USING ACCOUNT
          WITHIN MARKETING-AREA.

02        ACCOUNT           PIC X(6).
02        CUSTOMER-NAME     PIC X(30).
02        CUST-ADDRESS      PIC X(25).
02        CUST-CITY         PIC X(15).
02        CUST-STATE        PIC XX.
02        CUST-ZIP          PIC 9(5).
02        CUST-PHONE        PIC X(12).
02        CREDIT-STATUS     PIC 99.

RECORD NAME IS QTR-SALES-RECORD
          LOCATION MODE IS VIA SALES-SET
          WITHIN MARKETING-AREA.

02        QTR               PIC X(6).
02        SALES             PIC 9(6)V99.
```

Figure 4-2 (Cont.) The DML with COBOL: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

```
RECORD NAME IS PERFORMANCE-RECORD
          LOCATION MODE IS VIA PERFORMANCE-SET
          WITHIN MARKETING.

02        QTR                PIC X(6).
02        PREDICTION         PIC 9(7)V99.
02        PERFORMANCE        PIC 9(7)V99.

SET NAME IS COMMISSION-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS ALWAYS LAST
          OWNER IS SALESMAN-RECORD
          MEMBER IS QTR-COMMISSION-RECORD MAND AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME IS FIELD-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS ALWAYS LAST
          OWNER IS SALESMAN-RECORD
          MEMBER IS SALESFIELD-RECORD OPTIONAL AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME IS CUSTOMER-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS ALWAYS LAST
          OWNER IS SALESFIELD-RECORD
          MEMBER IS CUSTOMER-RECORD OPTIONAL AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME IS SALES-SET
          MODE IS CHAIN LINKED TO PRIOR
          OWNER IS ALWAYS LAST
          OWNER IS CUSTOMER-RECORD
          MEMBER IS QTR-SALES-RECORD MAND AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME IS PERFORMANCE-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS SORTED
          OWNER IS SALESFIELD-RECORD
          MEMBER IS PERFORMANCE-RECORD MAND AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SUB-SCHEMA NAME IS SUB-SCHEMA-1
PRIVACY LOCK IS SALEX.
AREA SECTION.
          COPY ALL AREAS.
```

Figure 4-2 (Cont.) The DML with COBOL: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

```
RECORD SECTION.
01        SALESMAN-RECORD.
01        QTR-COMMISSION-RECORD.
01        SALESFIELD-RECORD.
01        CUSTOMER-RECORD.
01        QTR-SALES-RECORD.
01        PERFORMANCE-RECORD.
SET SECTION.
          COPY ALL SETS.

END-SCHEMA.
```

Figure 4-2 (Cont.) The DML with COBOL: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

4.3.1 Example 1: Calculation of Salesmen Commissions and Bonuses

At the end of the first quarter, BARH Ltd. calculates the actual sales (i.e., performance) of each of its sales territories. This calculation is based on the sales made to each customer. Each salesman is to receive a commission equal to 12 percent of the sales made in his territory and a bonus of 5 percent on the amount of sales made over prediction. The program presented below performs these calculations.

Starting with the CUSTOMER-RECORD record, the program requests the quarter sales (stored in QTR-SALES-RECORD record), and adds them to the PERFORMANCE-RECORD record occurrence associated with the sales territory to which the customer belongs. After this is done for all the customers, the program selects the salesmen, one at a time, requests the prediction and performance for the respective sales territories, and calculates the commissions and bonuses to be paid. The commission and bonus earned by a salesman is then placed in a new QTR-COMMISSION-RECORD record, and the data base is accordingly modified. When all the salesmen have been processed, the program is finished.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXMPL1.
REMARKS.
     CALCULATES SALES FIELD PERFORMANCE BASED ON SALES TO CUSTOMERS
     IN THE MOST RECENT QUARTER AND SO MODIFIES DATA BASE. ALSO
     CALCULATES COMMISSION AND BONUS (IF ANY) TO BE PAID TO THE
     INDIVIDUAL SALESMEN ACCORDING TO THEIR SALES PERFORMANCE DURING
     THE QUARTER.

DATA DIVISION.
SCHEMA SECTION.                                    -(1)
     INVOKE SUB-SCHEMA-1 OF SCHEMA BARHEX          -(2)
         PRIVACY KEY COMPILE SALEX.

PROCEDURE DIVISION.
MAIN SECTION.
START.
     OPEN ALL USAGE-MODE IS PROTECTED UPDATE.      -(3)
     IF ERROR COUNT> 0,
         DISPLAY ERROR-STATUS. GO TO FINISH.       -(4)
```

```
ZERO-PROCEDURE.
     MOVE ZEROS TO PERFORMANCE.
     FIND LAST SALESFIELD-RECORD RECORD OF MARKETING-AREA AREA.
     IF ERROR-COUNT > 0, GO TO SALES-PROCEDURE.
LOOP-0.
     FIND LAST PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET.
     IF ERROR-COUNT > 0,
         DISPLAY ERROR-STATUS, GO TO FINISH.
     MODIFY PERFORMANCE-RECORD, PERFORMANCE.    -(5)
     FIND PRIOR SALESFIELD-RECORD RECORD OF MARKETING-AREA AREA.
     IF ERROR-COUNT  = 0, GO TO LOOP-0.  -(6)  -(7)

SALES-PROCEDURE.
     FIND FIRST CUSTOMER-RECORD RECORD OF MARKETING-AREA AREA.
LOOP-1.
     FIND LAST QTR-SALES-RECORD RECORD OF SALES-SET SET,
         SUPPRESS AREA CURRENCY UPDATES.  -(8)
     GET.                          -(9)
     FIND OWNER CUSTOMER-SET,                    -(10)
         SUPPRESS AREA CURRENCY UPDATES.
     FIND LAST PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET,
         SUPPRESS AREA CURRENCY UPDATES.
     IF ERROR-COUNT > 0,
         DISPLAY ERROR-STATUS, GO TO FINISH.
     ADD SALES TO PERFORMANCE.
     MODIFY.
     IF ERROR-COUNT > 0,
         DISPLAY ERROR-STATUS, GO TO FINISH.  -(11)
     FIND OWNER OF SALES-SET SET.
     FIND NEXT CUSTOMER-RECORD RECORD OF MARKETING-AREA AREA.
     IF ERROR-COUNT = 0, GO TO LOOP-1.

COMMISSION-PROCEDURE.
     FIND FIRST SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA.
     IF ERROR-COUNT > 0,
         DISPLAY ERROR-STATUS, GO TO FINISH.     -(12)
     CALL SUBPR USING PERFORMANCE-RECORD, QTR-COMMISSION-RECORD,
         SYSCOM,                            -(13)

          FINISH.
     CLOSE ALL.     -(14)
     STOP RUN.

IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPR.
REMARKS.
     SUBPROGRAM TO PERFORM THE CALCULATIONS FOR THE
     COMMISSIONS FOR THE SALESMEN.

DATA DIVISION.
SCHEMA SECTION.
     ACCESS SUB-SCHEMA-1 OF SCHEMA BARHEX     -(15)
         PRIVACY KEY COMPILE SALEX.
```

```
WORKING-STORAGE SECTION.
01    TEMP-SALES        PIC 9(7)V99,  USAGE COMP.
01    TEMP-COMMISSION   PIC 9(5)V99,  USAGE COMP.
01    TEMP-BONUS        PIC 9(5)V99,  USAGE COMP.

PROCEDURE DIVISION USING PERFORMANCE-RECORD, QTR-COMMISSION-RECORD,
     SYSCOM.                  -(16)
COMPT-SECTION.
LOOP-2.
     FIND NEXT RECORD OF FIELD-SET SET.     -(17)
     FIND LAST PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET.
     IF ERROR COUNT> 0,
         DISPLAY ERROR-STATUS, GO TO FINISH.
     GET.
     MOVE PERFORMANCE TO TEMP-SALES.
     COMPUTE TEMP-COMMISSION = TEMP-SALES * 0.12.
     MOVE ZEROS TO TEMP-BONUS.
     IF PREDICTION > PERFORMANCE, GO TO BOUNUS-CALCED.
     COMPUTE TEMP-BONUS = 0.05 * (PERFORMANCE - PREDICTION).
BONUS-CALCED.
     MOVE TEMP-COMMISSION TO COMMISSION.
     MOVE TEMP-BONUS TO BONUS.
     FIND LAST QTR-COMMISSION-RECORD RECORD OF COMMISSION-SET SET,
         SUPPRESS AREA CURRENCY UPDATES.     -(18)
     IF ERROR-COUNT > 0,
         DISPLAY ERROR-STATUS, GO TO FINISH.
     GET.
     MODIFY.
DONE-1.
     FIND OWNER COMMISSION-SET SET.
     FIND NEXT SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA.
     IF ERROR-COUNT = 0, GO TO LOOP-2.     -(7)
FINISH.
     EXIT PROGRAM.
```

(1) The SCHEMA SECTION must be included in the DATA DIVISION when DML statements are included in the program.

(2) An INVOKE or ACCESS statement must be the only statement in the SCHEMA SECTION.

(3) PROTECTED UPDATE (a simultaneous-update usage mode) permits other run-units to concurrently open the two areas to retrieve data, but not to update, until the CLOSE statement is executed in this program.

(4) If the execution of the OPEN statement fails, continuation of the run is not desired. Note that ERROR-COUNT is checked rather than ERROR-STATUS because it causes an integer compare rather than a character compare.

(5) The object of MODIFY is the current record of the run-unit. A GET statement is unnecessary here because the MODIFY explicitly names the only field it affects. Note that this is probably more careful than necessary because the "prediction program" should have set all new PERFORMANCE to 0.

(6) It is possible to search for the first (last) occurrence of a record type in an area, and then continue the search in the forward (reverse) direction; e.g., FIND PRIOR RECORD OF area-name AREA. FIND PRIOR should not be used to search for records in sets, however, unless the members are linked in the PRIOR direction.

(7) Since ERROR-COUNT is set to zero upon a successful FIND, this provides the test for continuation of looping using FINDs. The first nonzero ERROR-COUNT will occur when no record is found. A more definitive check would include testing for an ERROR-STATUS value of 0307.

(8) Area currency updating is suppressed so that the DBCS will not lose its place on subsequent NEXT OF AREA searches for CUSTOMER-RECORD. Refer to Chapter 3 for a description of FIND rec3.

(9) The object of GET is the current record of the run-unit. The GET moves the data-item fields within the current record into the appropriate UWA locations.

(10) SALESFIELD-RECORD record is the owner.

(11) If the FIND fails,terminate execution.

(12) If no salesman can be found, terminate execution.

(13) A call is made to a subprogram to compute the commissions. The call passes only the data needed and the System Communications Area (SYSCOM).

(14) Close all open areas before terminating the program. Failure to issue a CLOSE statement can cause some update activity for this program to not be reflected in the data base.

(15) An ACCESS statement is included in the SCHEMA SECTION of the subprogram so that the subprogram can reference data in the sub-schema invoked in the main program.

(16) The PROCEDURE DIVISION header contains the arguments that are passed to the subprogram.

(17) NEXT RECORD OF FIELD-SET gives the sales territory belonging to the current salesman.

(18) QTR-COMMISSION-RECORD must be made the current of run-unit in order to MODIFY it.

4.3.2 Example 2: Generation of First Quarter Salesman Commission and Bonus Report

BARH management wants a report on the performance of its salesmen in the most recent quarter, along with the amounts of commission and bonus to be paid. This program retrieves the information from the data base (as updated in the previous example) and writes the report using the COBOL Report Writer. The logic behind the retrieval in this case is quite simple: search for a salesman; find his commission and bonus from the most recent QTR-COMMISSION-RECORD; then retrieve the prediction and performance for his sales territory. When all this information is in the User Working Area, generate a line of the report.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXMPL2.
REMARKS.
     PREPARES A REPORT ON THE PERFORMANCE OF ALL BARH SALESMEN
     BY RETRIEVING THE DATA FROM THE DATA BASE WHICH WAS UPDATED BY
     EXMPL1.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT EXAMPLE-FILE; ASSIGN TO DSK; RECORDING MODE IS ASCII.

DATA DIVISION.
FILE SECTION.
FD  EXAMPLE-FILE;
         VALUE OF IDENTIFICATION IS "SALEMNRPT";
         REPORT IS EXAMPLE-REPORT.
01   EXAMPLE-RECORD          PIC X(72)
SCHEMA SECTION.              -(1)
INVOKE SUB-SCHEMA-1 OF SCHEMA BARHEX
     PRIVACY KEY COMPILE SALEX.
WORKING-STORAGE SECTION.
01   THIS-YEAR               PIC 9(4).

REPORT SECTION.
RD   EXAMPLE-REPORT, CONTROL FINAL.
01   TYPE PH; NEXT GROUP PLUS 2.
     02 LINE 3.
     03 COLUMN   1     PIC X(50);
 VALUE "FIRST QUARTER SALESMAN COMMISSION AND BONUS REPORT".
     03 COLUMN   52    PIC 9(4); SOURCE THIS-YEAR.
     03 COLUMN   65    PIC X(4); VALUE "PAGE".
     03 COLUMN   69    PIC ZZZ9; SOURCE PAGE-COUNTER.
 02 LINE 5.
     03 COLUMN   12    PIC X(8); VALUE "SALESMAN".
     03 COLUMN   32    PIC X(9); VALUE "PREDICTED".
     03 COLUMN   46    PIC X(6); VALUE "ACTUAL".
     03 COLUMN   54    PIC X(10); VALUE "COMMISSION".
     03 COLUMN   67    PIC X(5); VALUE "BONUS".

 02 LINE 6.
     03 COLUMN   34    PIC X(5); VALUE "SALES".
     03 COLUMN   46    PIC X(5); VALUE "SALES".
```

```
01    DETAIL-LINE; TYPE DE; NEXT GROUP PLUS 1.
      02 COLUMN 1  PIC X(30); SOURCE SALESMAN.
      02 COLUMN 31 PIC Z(6)9.99; SOURCE PREDICTION.
      02 COLUMN 43 PIC Z(6)9.99; SOURCE PERFORMANCE.
      02 COLUMN 55 PIC Z(4)9.99; SOURCE COMMISSION-1.
      02 COLUMN 65 PIC Z(4)9.99; SOURCE BONUS-1.

01    TYPE CF FINAL; LINE PLUS 1.
      02 COLUMN 2 PIC X(25);
          VALUE "** TOTAL FOR ALL SALESMAN".
      02 COLUMN 30 PIC Z(7)9.99; SUM PREDICTION.
      02 COLUMN 42 PIC Z(7)9.99; SUM PERFORMANCE.
      02 COLUMN 54 PIC Z(5)9.99; SUM COMMISSION-1.
      02 COLUMN 64 PIC Z(5)9.99; SUM BONUS-1.

PROCEDURE DIVISION.
MAIN SECTION.
START.
      OPEN ALL.     -(2)
      IF ERROR-COUNT > 0, GO TO FINISH.     -(3)
      DISPLAY "*".
      ACCEPT THIS-YEAR.
      OPEN OUTPUT EXAMPLE-FILE.
      INITIATE EXAMPLE-REPORT.

REPORT-PROCEDURE.
      FIND FIRST SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA.
LOOP-1.
      GET.     -(4)
      FIND LAST QTR-COMMISSION-RECORD RECORD OF
           COMMISSION-SET SET SUPPRESS AREA UPDATE.     -(5)
      IF ERROR-COUNT > 0, DISPLAY ERROR-STATUS, GO TO DONE-1.  -(6)
      GET.
      FIND NEXT RECORD OF FIELD-SET SET.
      IF ERROR-COUNT > 0, DISPLAY ERROR-STATUS, GO TO DONE-1.  -(6)
      GET.
      FIND LAST PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET
           SUPPRESS AREA.     -(5)
      GET.
      GENERATE DETAIL-LINE.
      FIND NEXT SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA.
      IF ERROR-COUNT = 0, GO TO LOOP-1     -(7)
DONE-1.
      TERMINATE EXAMPLE-REPORT.
      CLOSE EXAMPLE-FILE.

FINISH.
      CLOSE ALL.     -(8)
      STOP RUN.
```

(1) An INVOKE or ACCESS statement must be the only statement in the SCHEMA SECTION.

(2) With no USAGE-MODE specified, RETRIEVAL is assumed.

(3) If the areas cannot be opened, terminate execution.

(4) The object of GET is the current record of the run-unit.

(5) Note the use of the SUPPRESS phrase.

(6) If occurrences of these record types cannot be found, terminate execution.

(7) Continue looping until there are no more occurrences of SALESMAN-RECORD.

(8) Close the areas before terminating the program.

FIRST QUARTER SALESMAN COMMISSION AND BONUS REPORT 1975 PAGE 1

SALESMAN	PREDICTED SALES	ACTUAL SALES	COMMISSION	BONUS
ROBERT D. ACRES	450000.00	475789.17	57094.70	1289.45
WILLIAM J. AMBROSE	250000.00	242771.35	29132.56	0.00
JOSEPH T. ANDERSON	200000.00	202935.25	24352.23	146.76
KEVIN R. ANDERSON	225000.00	239447.00	28733.64	722.[illegible]
JAMES T. BARRON	100000.00	116534.58	13984.14	826.7[illegible]
R. CHARLES BLAKER	175000.00	178145.00	21377.40	157.25
JAMES D. BLOOM	155000.00	166444.80	19973.37	572.24
HAROLD T. BRENNEN	250000.00	240637.24	28876.46	0.00
RICHARD L. CAPPENTER	150000.00	166242.37	19949.08	812.11
HENRY G. CHOPAK	150000.00	169405.25	20328.63	970.26
JOHN F. COOPER	150000.00	174632.01	20955.84	1231.60
PATRICK R. FARMER	275000.00	278747.45	33449.69	187.37
ANDREW C. GATES	200000.00	206717.35	24806.08	335.[illegible]
CHARLES D. HARRINGTON	125000.00	142934.95	17152.19	896.[illegible]
LAWRENCE M. JAMESON	200000.00	228374.85	27404.98	1418.74
HANS K. JORGENSEN	175000.00	170724.58	20486.94	0.00
SAMUEL B. LEVIN	200000.00	187825.36	22539.04	0.00
JOHN L. LEWIS	300000.00	302147.46	36257.70	107.37
EDWARD S. MATTHEWS	205000.00	214052.31	25686.27	452.61
CHRISTOPHER W. NORTON	110000.00	114335.14	13720.21	216.75
JAMES T. OLSEN	230000.00	233924.99	28070.99	196.24
STEVEN G. SMITH	200000.00	192713.26	23125.59	0.00
MARK L. SMYTHE	175000.00	183224.97	21986.99	411.24
BARRY P. WILLIAMSON	130000.00	120625.17	14475.02	0.00
WILLIAM B. VANCKO	400000.00	413957.00	49674.84	697.85
** TOTAL FOR ALL SALESMAN	5180000.00	5363288.96	643594.58	11649.51

4.3.3 Example 3: Generation of Prediction Accuracy Report

In addition to the report on the performance of the salesmen, BARH requires a report on the accuracy of the predictions of each salesman. The report gives the percent of acceptable variance between the prediction and the actual performance, the number of quarters in the sample, and the number of times the prediction was too high or too low. The data for the example is the same as that for the previous example; the data base is updated as in the first example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXMPL3.
REMARKS.
     PREPARES A REPORT ON THE PREDICTION ACCURACY OF ALL BARH
     SALESMEN BY RETRIEVING DATA FROM THE BARHEX DATA BASE.


ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT EXAMP-FILE, ASSIGN TO DSK, RECORDING MODE IS ASCII.

DATA DIVISION.
FILE SECTION.
FD EXAMP-FILE,
         VALUE OF IDENTIFICATION IS "PREDICRPT",
         REPORT IS EXAMP-REPORT.
01 EXAMP-RECORD     PIC X(72).

SCHEMA SECTION.
     INVOKE SUB-SCHEMA-1 OF SCHEMA BARHEX
         PRIVACY KEY COMPILE SALEX.

WORKING-STORAGE SECTION.
77 VARIANCE  PIC 99.
77 QUARTERS  PIC 999.
77 TOO-HIGH  PIC 999.
77 TOO-LOW   PIC 999.
77 QTR-COUNT PIC 999.
77 DIF       PIC 9(7)V99.
77 ALLOW-D   PIC 9(7)V99.
REPORT SECTION.
RD EXAMP-REPORT, CONTROL FINAL.
01 TYPE PH, NEXT GROUP PLUS 2.
     02 LINE 3.
         03 COLUMN 1 PIC X(50),
     VALUE "SALESMAN PREDICTION ACCURACY REPORT".
         03 COLUMN 65 PIC X(4), VALUE "PAGE".
         03 COLUMN 69 ZZZ9, SOURCE PAGE-COUNTER.
     02 LINE 4.
         03 COLUMN 12 PIC X(25), VALUE "% ACCEPTABLE VARIANCE =".
         03 COLUMN 40 PIC 99,     SOURCE VARIANCE.
         03 COLUMN 45 PIC X(20), VALUE "QUARTERS IN SAMPLE =".
         03 COLUMN 68 PIC 999, SOURCE QUARTERS.
```

```
     02 LINE 6.
          03 COLUMN 12 PIC X(8), VALUE "SALESMAN".
          03 COLUMN 32 PIC X(15), VALUE "SIGNIFICANTLY".
          03 COLUMN 50 PIC X(15), VALUE "SIGNIFICANTLY".
          03 COLUMN 67 PIC X(5), VALUE "TOTAL".
     02 LINE 7.
          03 COLUMN 32 PIC X(8), VALUE "TOO HIGH".
          03 COLUMN 50 PIC X(8), VALUE "TOO LOW".

01   DETAIL-LINE, TYPE DE, NEXT GROUP PLUS 1.
     02 COLUMN 1 PIC X(30), SOURCE SALESMAN.

     02 COLUMN 32 PIC 999, SOURCE TOO-HIGH.
          02 COLUMN 50 PIC 999, SOURCE TOO-LOW.
          02 COLUMN 68 PIC 999, SOURCE QTR-COUNT.

PROCEDURE DIVISION.
MAIN SECTION.
START.
     OPEN ALL.
     IF ERROR-COUNT > 0, GO TO FINISH.
     DISPLAY "% VARIANCE?".
     ACCEPT VARIANCE.
     DISPLAY "SAMPLE SIZE?".
     ACCEPT QUARTERS.
     OPEN OUTPUT EXAMP-FILE.
     INITIATE EXAMP-REPORT.

REPORT-PROCEDURE.
     FIND FIRST SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA.
LOOP-1.
     MOVE ZEROS TO TOO-LOW, TOO-HIGH, QTR-COUNT.
     GET.
     FIND NEXT RECORD OF FIELD-SET SET, SUPPRESS AREA UPDATES.
     IF ERROR-COUNT > 0, DISPLAY ERROR-STATUS, GO TO DONE-1.
     GET.
IN-1.
     FIND PRIOR PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET,
          SUPPRESS AREA.
     IF ERROR-COUNT NOT = 0, GO TO GEN-1.
     GET.
     ADD 1 TO QTR-COUNT.
     COMPUTE DIF = PERFORMANCE - PREDICTION.
     COMPUTE ALLOW-D = PREDICTION * (VARIANCE/100).
     IF DIF > ALLOW-D ADD 1 TO TOO-LOW.
     IF -DIF > ALLOW-D ADD 1 TO TOO-HIGH.
     IF QTR-COUNT < QUARTERS GO TO IN-1.
GEN-1.
     GENERATE DETAIL-LINE.
     FIND NEXT SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA.
     IF ERROR-COUNT = 0, GO TO LOOP-1.
```

```
DONE-1.
     TERMINATE EXAMP-REPORT.
     CLOSE EXAMP-FILE.

FINISH.
     CLOSE ALL.
     STOP RUN.
```

CHAPTER 5
USING THE DML IN FORTRAN PROGRAMS

You can embed DBMS Data Manipulation Language statements in a FORTRAN program along with FORTRAN statements. Since the DML is not FORTRAN-oriented, you must run the FORTRAN source program through a preprocessor before compiling it. Once the program has passed through the preprocessor, you can compile with FORTRAN-20, and execute with FOROTS as you would any other FORTRAN program.

5.1 BUILDING A FORTRAN-DML PROGRAM

Creating an executable application program using the DML with FORTRAN involves a series of steps. These are illustrated schematically in Figure 5-1; they involve

1. Creating a new program (or using an existing program)
2. Running the FORDML preprocessor
3. Compiling the resulting FORTRAN program
4. Loading the relocatable object code
5. Optionally saving the loaded object code
6. Executing the run-unit.

5.2 PLACING DML STATEMENTS WITHIN FORTRAN

You must use the DML statements according to the conventions and rules discussed in Chapter 3. FORTRAN rules do not apply to them. When using the DML with FORTRAN, observe the following rules:

1. Statements can start and end in any column. The last non-blank character in a statement must be a period (.). FORTRAN labels can appear on statements and can also start in any column.
2. No line in a DML statement can contain all or part of another DML or FORTRAN statement.
3. There is no specific limit to the length of a line. The only limitation on statement length is that a statement cannot contain more than 120 symbols. (For example, an identifier is one symbol; a keyword is one symbol; and a period is one symbol.)
4. Commas (,) and semicolons (;) are completely equivalent to spaces.
5. Any line in a DML statement can be followed by a comment. A comment is preceded by an exclamation point (!) and is terminated by a line termination character (line feed, form feed, vertical tab).
6. For the preprocessor to recognize that a line in a source program contains a DML statement, the line must be preceded by a line containing:

   *<blanks> DBMS

   The blank characters are null, tab, space, or backspace.
7. As with FORTRAN, identifiers and keywords are treated as upper case. All other characters are output exactly as they are input. Blanks cannot be freely interspersed within symbols because a blank is a symbol terminator in DML statements. For example, "A B" is treated as "A" "B" within a DML statement whereas it is treated as "AB" in a FORTRAN statement.

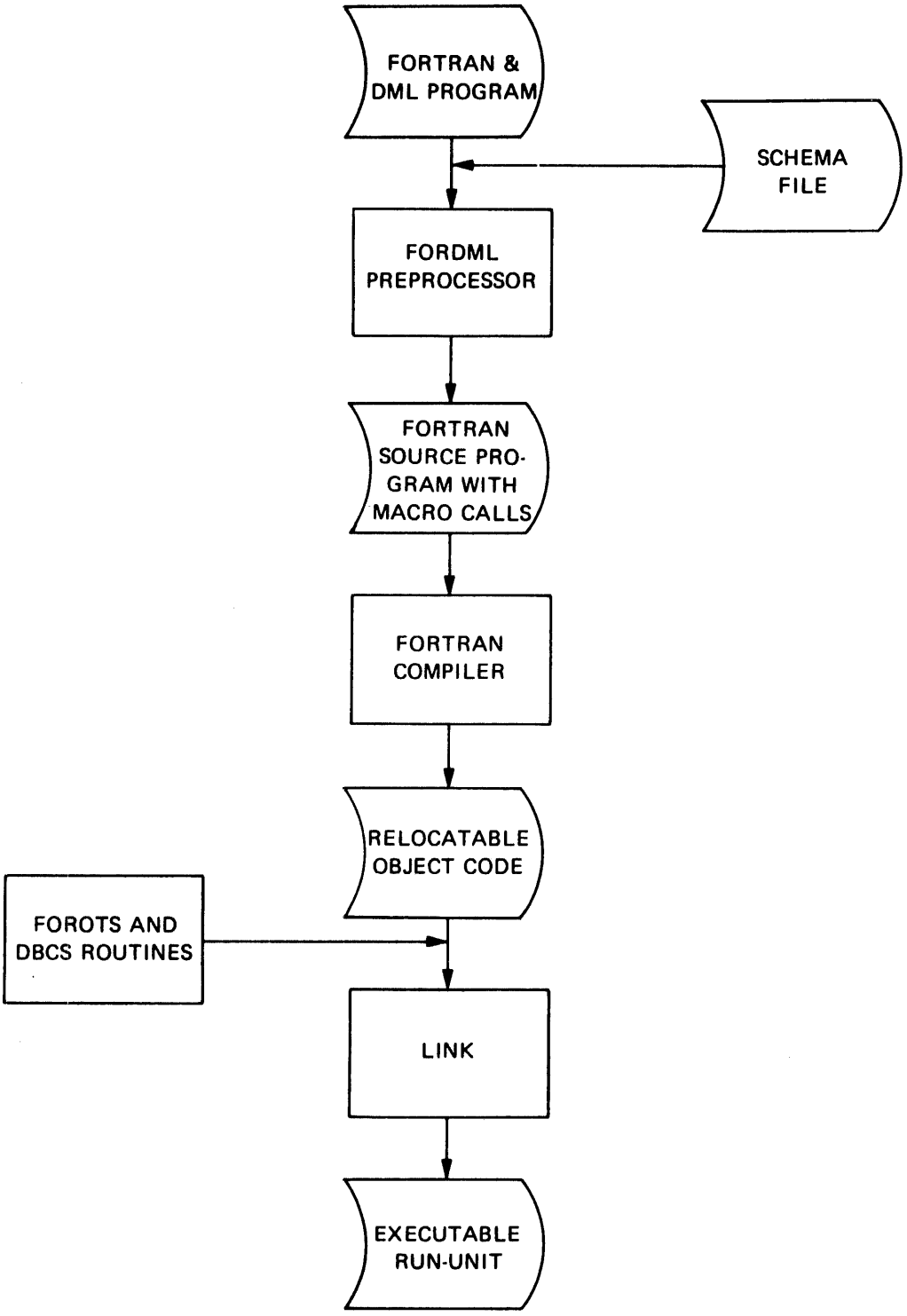


Figure 5-1 Program-Building Process for FORTRAN with DML

5.2.1 The INVOKE Statement

Place the INVOKE statement such that it follows the last declarative statement and precedes the first executable statement in a program-unit. When the INVOKE statement is processed, the preprocessor puts the data descriptions from the sub-schema into the FORTRAN program in the form of type statements (e.g., INTEGER, REAL). The schema data descriptions are in a host-language independent format; the preprocessor converts the data descriptions according to the definitions shown in Appendix C.

The data-names in a schema description are COBOL-oriented rather than FORTRAN-oriented. They can be up to 30 characters in length and can contain hyphens, while FORTRAN allows names up to six characters only and does not allow hyphens. The Data Base Administrator can define FORTRAN pseudonyms, however, to make data-names consistent with FORTRAN usage.

Translation of the names from the schema to the FORTRAN program follows the rules described below. Note that these rules apply only to data-names, and not to record, set, and area names; the latter names can never appear outside of a DML statement.

1. If a FORTRAN pseudonym is present, it is used as the data-name.
2. If a pseudonym is not present but the data-name from the schema contains six characters or less and no hyphens, the data-name from the schema is used as the data-name in the program.
3. If neither of the above is true, storage is allocated for the data-name, but neither a data typing statement nor a referenceable name is created. Instead, each name is placed in an array called UNDEF. The data-items placed in the UNDEF array cannot be referenced directly either by a DML or a FORTRAN statement.

The preprocessor creates the User Working Area (UWA) for each program when the INVOKE statement is processed. The UWA contains the data descriptions and the following special registers: ARNAM, RECNAM, ERSTAT, ERSET, ERREC, ERAREA, and ERCNT. They are placed in the program in the form:

```
INTEGER SYSCOM(33), ERCNT, ERSTAT
INTEGER ERAREA(6), ERREC(6), ERSET(6), RECNAM(6), ARNAM(6)
EQUIVALENCE(SYSCOM(1)ARNAM),
1(SYSCOM(7),RECNAM),
1(SYSCOM(13), ERSTAT),
1(SYSCOM(14), ERSET),
1(SYSCOM(20), ERREC),
1(SYSCOM(26), ERAREA),
1(SYSCOM(32), ERCNT)
```

ARNAM and RECNAM are used to store the last area name and record name processed by a MOVE STATUS statement. The other registers are used

1. To keep you informed of the occurrence of an exception during execution of a DML statement (ERSTAT, ERCNT) and
2. To give you information relevant to the occurrence of an exception (ERSET, ERREC, ERAREA). (Refer also to Section 3.2.1.)

For example, ERSTAT contains the statement code and exception code; ERSET, ERREC, and ERAREA contain the name of the set, record, and area, respectively, in which the exception occurs (if these are known to DBCS); ERCNT contains 1 if an exception occurs and 0 if none occurs.

The data-base descriptions are made global and are declared in a named common block. The name of the common block is the name of the sub-schema (the first six characters). The fact that the data descriptions appear in a common block allows you to reference the data from multiple program-units in a direct manner.

5.2.2 The ACCESS Statement

Use the ACCESS statement in a subprogram when you want that subprogram to access the data in a sub-schema previously invoked in another program-unit. When the subprogram is called from the program-unit containing an INVOKE statement or another ACCESS statement, the calling program-unit can call with no data-base arguments (unlike COBOL) because the subprogram references the entire UWA via the common block that contains the UWA for the accessed sub-schema.

Place the ACCESS statement (like the INVOKE statement) such that it follows the last declarative statement and precedes the first executable statement in a subprogram. Only one ACCESS statement can appear in a subprogram; and it cannot appear in a subprogram with an INVOKE statement. These restrictions are applied because only one sub-schema can be referenced in a given program-unit.

5.2.3 Other DML Statements

Place the DML statements CLOSE, DELETE, FIND, GET, INSERT, MODIFY, MOVE, STATUS, OPEN, REMOVE, and STORE anywhere in the executable portion of the program. If you use the END statement, place it at the end of the program-unit. Place the OPEN statement such that it is the first DML statement executed; the other DML statements will not execute successfully unless the area (or areas) to be used are open.

5.3 RUNNING THE PREPROCESSOR, FORDML

Before compiling your FORTRAN program containing DML statements, run it through FORDML, the DML preprocessor. The command string to run FORDML is:

```
.R FORDML
*[output spec=]input spec[/switch]
```

The output specification contains the device, filename, extension, and directory for the processed file. The default for the filename is that of the input file. The default extension is .FOR. The input specification is the device, filename, extension, and directory of the file to be processed. You must specify the input filename. The default extension is .FML. The default device for both the input and output files is DSK; the default directory for both is the current path.

The switches you can specify are as follows: /[NO]VIEW and /UNFLAGGED. The former (/[NO]VIEW) has two positions; /VIEW, which causes FORDML to allow generation of an expanded listing of every INCLUDE statement generated by an INVOKE statement and /NOVIEW, which causes FORDML to suppress expanded listings of INCLUDE statements. The latter (/UNFLAGGED) implies your program conforms to the following: no line other than the last line of a DML statement contains a period as its last non-blank character; however, a line can end with a remark. In effect, single-line DML statements need not be preceded by a line containing *DBMS.

The following three examples show the command string. All three examples are equivalent.

```
.R FORDML
*TST1
```

The name of the input file is TST1; its extension is assumed to be .FML. The output file is named TST1.FOR.

```
.R FORDML
*TST1=TST1
```

The filenames and extensions are the same as above.

```
.R FORDML
*TST1.FOR=TST1.FML
```

The filenames and extensions are as described in the first example.

You can use wildcarding in both the input- and output-file specifications. The wildcard characters are asterisk (*) and question mark (?). Asterisks replace a filename or extension, question marks replace single characters. The following rules apply:

1. If both specifications are wild, one output file will be generated for each input file found.
2. If the output specification is not wild and the input specification is, the translated input files are concatenated to form one output file. FORDML assumes that an input file contains an integral number of program-units. Do not therefore divide program-units between files.
3. When the output specification is omitted and the input specification is wild, both specifications are treated as wild.

5.4 EXAMPLE USING DML IN FORTRAN PROGRAMS

The example presented on the following pages illustrates use of DML statements in FORTRAN programs. The program is preceded by a description of the example and is followed by numbered explanatory notes. These notes refer to statements, or segments of the program, which are denoted by a "note marker" in the form:

```
OPEN ALL    -(1)
```

where -(1) is the marker.

The example concerns the same sub-schema of the schema BARHEX that was used in the COBOL examples in Chapter 4. It performs the same computations as the first COBOL example. Compare the examples to see how the same problem is solved in COBOL and in FORTRAN, both using the DML. Note that FORTRAN pseudonyms have been added to the schema where necessary.

Figure 5-2 shows the schema BARHEX and the sub-schema SUB-SCHEMA-1. Refer to the schematic notation in Figure 1-5 in Chapter 1, for assistance in visualizing the relationships among the records in the schema.

```
RECORDS-PER-PAGE 36.

ASSIGN PERSONNEL-AREA TO FILE1
FIRST PAGE IS 1
LAST PAGE IS 21
PAGE SIZE IS 256 WORDS.

ASSIGN MARKETING-AREA TO FILE2
FIRST PAGE IS 101
LAST PAGE IS 201
PAGE SIZE IS 256 WORDS.
```

Figure 5-2 The DML with FORTRAN: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

```
SCHEMA NAME IS BARHEX.

AREA NAME IS PERSONNEL-AREA.
AREA NAME IS MARKETING-AREA.

RECORD NAME IS SALESMAN-RECORD
          LOCATION MODE IS DIRECT IDENT1
          WITHIN PERSONNEL-AREA.

02        SALESMAN          PIC X(30).
02        HOME-ADDRESS      PIC X(25).
02        HOME-CITY         PIC X(15).
02        HOME-STATE        PIC XX.
02        HOME-ZIP          PIC 9(5).
02        HOME-PHONE        PIC X(12).
02        SS-NUMBER         PIC X(11).
02        BASE-SALARY       PIC 9(5)V99.
02        HIRING-DATE       PIC X(8).

RECORD NAME IS QTR-COMMISSION-RECORD
          LOCATION MODE IS VIA COMMISSION-SET
          WITHIN PERSONNEL-AREA.

02        QTR                       PIC X(6).
02        COMMISSION%COMMIS         PIC 9(5)V99.
02        BONUS                     PIC 9(5)V99.

RECORD NAME IS SALESFIELD-RECORD
          LOCATION MODE IS DIRECT IDENT2
          WITHIN MARKETING-AREA.

02        FIELD-NUMBER      PIC 9999.
02        FIELD-LOCALE      PIC X(30).

RECORD NAME IS CUSTOMER-RECORD.
          LOCATION MODE IS CALC USING ACCOUNT
          WITHIN MARKETING-AREA.

02        ACCOUNT           PIC X(6).
02        CUSTOMER-NAME     PIC X(30).
02        CUST-ADDRESS      PIC X(25).
02        CUST-CITY         PIC X(15).
02        CUST-STATE        PIC XX.
02        CUST-ZIP          PIC 9(5).
02        CUST-PHONE        PIC X(12).
02        CREDIT-STATUS     PIC 99.

RECORD NAME IS QTR-SALES-RECORD
          LOCATION MODE IS VIA SALES-SET
          WITHIN MARKETING-AREA.
```

Figure 5-2 (Cont.) The DML with FORTRAN: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

```
02        QTR                PIC X(6).
02        SALES              PIC 9(6)V99.

RECORD NAME IS PERFORMANCE-RECORD
          LOCATION MODE IS VIA PERFORMANCE-SET.
          WITHIN MARKETING-AREA.

02        QTR                      PIC X(6).
02        PREDICTION$PREDIC        PIC 9(7)V99.
02        PERFORMANCE$PERFOR       PIC 9(7)V99.

SET NAME IS COMMISSION-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS ALWAYS LAST
          OWNER IS SALESMAN-RECORD
          MEMBER IS QTR-COMMISSION-RECORD MAND AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME IS FIELD-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS ALWAYS LAST
          OWNER IS SLAESMAN-RECORD
          MEMBER IS SALESFIELD-RECORD OPTIONAL AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME CUSTOMER-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS ALWAYS LAST
          OWNER IS SALESFIELD-RECORD
          MEMBER IS CUSTOMER-RECORD OPTIONAL AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME IS SALES-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS ALWAYS LAST
          OWNER IS CUSTOMER-RECORD
          MEMBER IS QTR-SALES-RECORD MAND AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

SET NAME IS PERFORMANCE-SET
          MODE IS CHAIN LINKED TO PRIOR
          ORDER IS SORTED
          OWNER IS SALESFIELD-RECORD
          MEMBER IS PERFORMANCE-RECORD MAND AUTO LINKED TO OWNER
          SET SELECTION CURRENT.

          SUB-SCHEMA NAME IS SUB-SCHEMA-1
          PRIVACY LOCK IS SALEX.
          AREA SECTION.
               COPY ALL AREAS.
```

Figure 5-2 (Cont.) The DML with FORTRAN: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

```
RECORD SECTION.
01     SALESMAN-RECORD.
01     QTR-COMMISSION-RECORD.
01     SALESFIELD-RECORD.
01     CUSTOMER-RECORD.
01     QTR-SALES-RECORD.
01     PERFORMANCE-RECORD.
SET SECTION.
     COPY ALL SETS.

END-SCHEMA.
```

Figure 5-2 (Cont.) The DML with FORTRAN: Example Schema BARHEX and Sub-Schema SUB-SCHEMA-1

At the end of the first quarter, BARH Ltd. calculates the actual sales (i.e., performance) of each of its sales territories. This calculation is based on the sales made to each customer. The company also calculates the commissions and bonuses for the salesmen. Each salesman is to receive a commission equal to 12 percent of the sales made in his territory and a bonus of 5 percent on the amount of sales made over prediction. The program presented below performs these calculations.

Starting with the CUSTOMER-RECORD record, the program requests the quarter sales (stored in QTR-SALES-RECORD record), and adds them to the PERFORMANCE-RECORD record occurrence associated with the sales territory to which the customer belongs. After this is done for all the customers, the program selects the salesmen, one at a time; requests the prediction and performance for the respective sales territories; and calculates the commissions and bonuses to be paid from these data items. The commission and bonus earned by each salesman is then placed in a new QTR-COMMISSION-RECORD record, and the data base is accordingly modified. When all the salesmen have been processed, the program is finished.

The program assumes that a function exists called CONVRT that converts a numeric SIXBIT item to a real number and a subprogram exists called NUMER6 that converts a real number to a numeric SIXBIT item.

```
C     CALCULATES SALES FIELD PERFORMANCE BASED ON SALES TO
C     CUSTOMER IN THE MOST RECENT QUARTER AND SO MODIFIES
C     DATA BASE. ALSO CALCULATES COMMISSION AND BONUS (IF
C     ANY) TO BE PAID TO THE INDIVIDUAL SALESMEN ACCORDING
C     TO THEIR SALES PERFORMANCE DURING THE QUARTER.

*     DRMS    -(1)
      INVOKE SUB-SHCEMA-1 OF SCHEMA BARHEX    -(2)
      PRIVACY KEY COMPILE SALEX.
*     DRMS
      OPEN ALL USAGE-MODE IS PROTECTED UPDATE.    -(3)
      IF (ERSTAT .GT. 0) GO TO 88    -(4)
      CALL NUMER6(PERFOR, 0.)
*     DRMS
      FIND LAST SALESFIELD-RECORD RECORD OF MARKETING-AREA AREA.
      IF (ERSTAT .GT. 0) GO TO 12
*     DRMS
11    FIND LAST PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET.
      IF (ERSTAT .GT. 0) GO TO 88
*     DRMS
      MODIFY PERFORMANCE-RECORD, PERFOR.     -(5)
*     DRMS
      FIND PRIOR SALESFIELD-RECORD RECORD OF MARKETING-AREA AREA.
      IF (ERSTAT .EQ. 0) GO TO 11    -(6) -(7)
```

```
*     DBMS
12    FIND FIRST CUSTOMER-RECORD RECORD OF MARKETING-AREA AREA.
*     DBMS
121   FIND LAST QTR-SALES-RECORD RECORD OF SALES-SET SET
           SUPPRESS AREA CURRENCY UPDATES.   -(8)
*     DBMS
      GET.   -(9)
*     DBMS
      FIND OWNER CUSTOMER-SET,   -(10)
           SUPPRESS AREA CURRENCY UPDATES.
*     DBMS
      FIND LAST PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET.
           SUPPRESS AREA CURRENCTY UPDATES.
      IF (ERSTAT .GT. 0) GO TO 88
      CALL NUMER6 (PERFOR, CONVRT (PERFOR) + CONVRT (SALES))
*     DBMS
      MODIFY.
      IF (ERSTAT .GT. 0) GO TO 88   -(11)
*     DBMS
      FIND OWNER OF SALES-SET SET.
*     DBMS
      FIND NEXT CUSTOMER-RECORD RECORD OF MARKETING-AREA AREA.
      IF (ERSTAT .EQ. 0) GO TO 121
*     DBMS
      FIND FIRST SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA.
      IF (ERSTAT .GT. 0) GO TO 88   -(12)
      CALL SUBPR   -(13)
88    TYPE 101 ERSTAT
101   FORMAT ('?ERROR-STATUS:',I4)
*     DBMS
      CLOSE ALL.   -(14)
*     DBMS
      END EXAMPL.

      SUBROUTINE SUBPR
C     SUBPROGRAM TO PERFORM THE CALCULATIONS FOR THE
C     COMMISSIONS FOR THE SALESMEN.
      REAL TSALES, TCOMM, TBONUS
*     DBMS
      ACCESS SUB-SCHEMA-1 OF SCHEMA BARHEX   -(15)
           PRIVACY KEY COMPILE SALEX.
*     DBMS
      FIND NEXT RECORD OF FIELD-SET SET.   -(16)
*     DBMS
      FIND LAST PERFORMANCE-RECORD RECORD OF PERFORMANCE-SET SET.
      IF (ERSTAT .GT. 0) GO TO 88
*     DBMS
      GET.
      TSALES = CONVRT (PERFOR)
      TCOMM = TSALES * 0.12
      TBONUS = 0
      TEMP = CONVRT (PREDIC) - CONVRT (PERFOR)
      IF(TEMP .GT. 0) GO TO 131
      TBONUS = TEMP * .05
```

```
131   CALL NUMER6 (COMMIS, TCOMM)
      CALL NUMER6 (BONUS, TBONUS)
*     DBMS
      FIND LAST QTR-COMMISSION RECORD RECORD OF COMMISSION-SET SET
           SUPPRESS AREA CURRENCY UPDATES,   -(17)
      IF (ERSTAT .GT.0) GO TO 88
*     DBMS
      MODIFY,
*     DBMS
      FIND OWNER COMMISSION-SET SET,
*     DBMS
      FIND NEXT SALESMAN-RECORD RECORD OF PERSONNEL-AREA AREA,
      IF (ERSTAT .EQ. 0) GO TO 13    -(7)
88    TYPE 101, ERSTAT
101   ('?ERROR-STATUS:',I4)
      RETURN
*     DBMS
      END SUBPR.
```

(1) A line containing * DBMS must precede each DML statement in the program.

(2) The INVOKE statement must precede the first executable statement in the program.

(3) PROTECTED UPDATE permits other run-units to concurrently open the two areas to retrieve data, but not to update, until the CLOSE statement is executed in this program.

(4) If the execution of the OPEN statement fails, continuation of the run is not desired.

(5) The object of MODIFY is the current record of the run-unit. A GET statement is unnecessary here because the MODIFY explicity names the only field it affects. Note that this is probably more care than necessary because the "prediction program" should have set all new PERFORs to 0.

(6) You can always search for the first (last) occurrence of a record type in an area, and then continue the search in the forward (reverse) direction (e.g., FIND PRIOR RECORD OF area-name AREA) FIND PRIOR should not be used, however, to search for records in sets unless the number records are linked to PRIOR.

(7) Since ERSTAT is set to zero upon a successful FIND, this provides the test for continuation of looping using FINDs. The first nonzero ERSTAT will occur when no record is found. A more definitive check would include testing for an ERSTAT value of 0307.

(8) Area currency updating is suppressed so that the DBCS will not lose its place on subsequent NEXT OF AREA searches for CUSTOMER-RECORD.

(9) The object of GET is the current record of the run-unit. The GET moves the data-item fields within the current record into the appropriate UWA locations.

(10) SALESFIELD-RECORD record is the owner.

(11) If the FIND fails, execution should be terminated.

(12) If no salesman can be found, execution should be terminated.

(13) A call is made to a subprogram to compute the commissions. The call need not pass arguments because the data from the data base and SYSCOM are in COMMON.

(14) Close all open areas before terminating the program. Failure to issue a CLOSE statement might cause some update activity for this program not to be reflected in the data base.

(15) An ACCESS statement is included in the subprogram so that the subprogram can reference data in the sub-schema invoked in the main program.

(16) NEXT RECORD OF FIELD-SET SET gives the sales territory belonging to the current salesman.

(17) QTR-COMMISSION-RECORD must be made the current of run-unit in order to MODIFY it.

APPENDIX A

RESERVED WORDS AND USER-REFERENCABLE DBCS NAMES

This appendix lists DBMS reserved words and routine names. You cannot use these terms to name your own routines or other items in your programs.

A.1 RESERVED WORDS

The following words are reserved in DBMS, along with their abbreviations, which are enclosed in parentheses. You cannot use these words as user-created names in any DML statements. Refer to the *DECsystem-10 COBOL Programmer's Reference Manual* for COBOL reserved words. Those words preceded with an asterisk refer to COBOL only; those preceded by two asterisks refer to FORTRAN only.

-A-

ACCESS
AFTER
ALIAS
ALL
ALLOWED
ALWAYS
ARE
AREA
AREA-ID
*AREA-NAME
**ARNAM
ASCENDING
AUTOMATIC

-B-

BACKUP
BEFORE
BINARY (BIN)
BIT
BY

-C-

CALC
CALL
CHAIN
CLOSE
COMPILE
COMPLEX
CURRENT

-D-

DATABASE-KEY (DBKEY)
DECIMAL (DEC)
DELETE
DESCENDING (DESC)
DIRECT
DISPLAY
DUPLICATE
DUPLICATES
DYNAMIC

-E-

ELSE
EMPTY
ENCODING
**END
**ERAREA
**ERCNT
**ERREC
*ERROR-AREA
*ERROR-COUNT
*ERROR-RECORD
*ERROR-SET
*ERROR-STATUS
**ERSER
**ERSTAT
EXCLUSIVE

-F-

FIND
FIRST
FIXED
FLOAT
FOR
FROM

-G-

GET

-I-

IF
IMAGES
IN
INDEX
INDEXED
INSERT
INTO
INVOKE
IS

-K-

KEY

-L-

LAST
LINKED
LOCATION
LOCK

-M-

MANDATORY
MANUAL
MEMBER
MEMBERS
MODE
MODIFY
MOVE

-N-

NEXT
NOT

-O-

OCCURRENCE
OCCURS
OF
ON
ONLY
OPEN
OPTIONAL
ORDER
OWNER

-P-

PICTURE (PIC)
PRIOR
PRIVACY
PROTECTED

-R-

RANGE
REAL
**RECNNAM
RECORD
*RECORD-NAME
REMOVE
RETRIEVAL
RUN-UNIT

-S-

SCHEMA
SEARCH
SELECTION
SELECTIVE
*SENTENCE
SET
SETS
SORTED
STATUS
STORE
SUB-SCHEMA
SUPPRESS

-T-

TEMPORARY
THRU
TIMES
TO

-U-

**UNDEF
UPDATE
UPDATES
USAGE
USAGE-MODE
USE
USING

-V-

VIA

-W-

WITHIN

A.2 USER-REFERENCABLE DBCS NAMES

This section identifies the DBCS routine-names you can use in explicit calls and informs you which names you cannot use as names in your own programs. Table A-1 lists the DBMS keywords and numerical values each has been assigned. Use these values instead of the keywords themselves in your explicit calls to DBCS entry points.

Table A-1 DBMS Keywords and Assigned Values

Keyword	Value
ONLY	-10
SELECTIVE	-11
FIRST	-12
LAST	-13
PRIOR	-14
NEXT	-15
DUPLICATES	-16
ALL	-17
AREA	-18
RECORD	-19
SET	-20

Table A-1 (Cont.) DBMS Keywords and Assigned Values

Keyword	Value
UPDATE	-21
RETRIEVAL	-22
RUNUNIT	-23
PROTECTED	-24
EXCLUSIVE	-25
RESERVED	-26
RESERVED	-27
JOURNAL	-28

Table A-2 lists the DBCS entry points and shows the arguments you can use when accessing them. The asterisks identify a synonym for the name immediately preceding. You can replace the keywords in the argument list with the negative values shown in Table A-1. Using this facility allows you to replace DML statements that you would otherwise redundantly code throughout your program with a single generic call. You can then provide different values for the variable (in your argument list) at run-time (for example, using the ACCEPT/DISPLAY commands).

You may, for example, want to find the next record of each of five sets. The generic call in FORTRAN would look this way:

```
CALL FIND3 (-15, 0, CURSET, -20)
```

The generic call in COBOL would look this way:

```
ENTER MACRO FIND3 USING -15, 0, CURSET, -20.
```

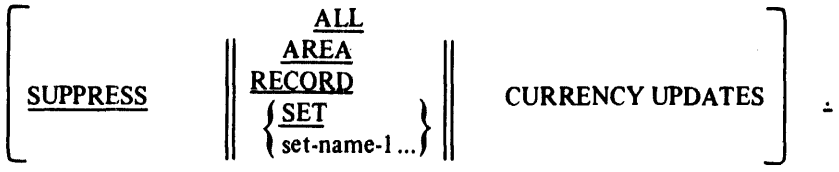
Table A-2 DBCS Entry Points and Arguments

CLOSED	(ALL)
CLOSED	(AREA, area-list)
DELETR	(0 or SELECT or ALL or ONLY)
FIND1	(record or 0, USING-value)
FIND2	(set or 0, currency, currency-keyword)
FIND3	(relative, record or 0, area or set, AREA or SET)
FIND0	(integer, record or 0, area or set, AREA or SET)
FIND4	(set)
FIND5	(DUPLICATES or 0, record)
GETS	(record [,data-list]) or (0)
*GET	
INSERT	(record or 0, set-list)
*INSRT	
MODIF	(same-as-GET)
*MODIFY	
MOVEC	(currency, currency-keyword, result)
OPEND	(RETR or UPDATE, 0 or PROT or EXCL, privacy-key, ALL or area-list)
REMOVE	(same-as-INSERT)
*REMOV	

Table A-2 (Cont.) DBCS Entry Points and Arguments

STORE	(record)
*STORED	
SBIND	(schema, edit, subschema, ss-mask, SYSCOM-address)
BIND	(record,data-address-list)
EBIND	(0,DBMS-NULL)

In addition, the FINDs and STOREs can have appended a SUPPRESS list as follows:



The remaining DBCS user-referencable routines are listed below.

SETDB	UNSET	SAVESS	JMNAME
JMAFT	JMBEF	JMBOTH	JMNONE
JBTRAN	JSTRAN	JETRAN	JRSYNC
EMPTY	*SETCON	MEMBER	*RECMEM
OWNER	*RECOWN	TENANT	*RECMO
STATS			

APPENDIX B

EXCEPTION CONDITION CODES AND ERROR MESSAGES

This appendix lists and discusses DBMS exception condition codes. It also lists (1) the DBCS run-time messages (2) the COBOL compiler error messages that can occur during compilation of your COBOL-DML program, and (3) the FORDML preprocessor error messages that can occur during preprocessing of your FORTRAN-DML program.

B.1 EXCEPTION CONDITION CODES

Exception handling in DBMS has been discussed in Section 3.2. Table B-1, which is a duplicate of Table 3-1, is repeated here so that you can easily associate the statement-function codes with the exception condition codes listed and described in Table B-2.

Table B-1
DML-Statement-Associated Functions and Codes

Code	Statement
00	HOST
01	CLOSE
02	DELETE
03	FIND
05	GET
07	INSERT
08	MODIFY
09	OPEN
11	REMOVE
12	STORE
15	BIND
16	CALL

Table B-2
Exception Condition Codes

Code	Condition
00	A warning. Compile-time and run-time versions of schema file differ. If a "real" exception occurs during binding, however, DBCS always returns the code of that exception. To indicate that exception 00 has occurred as well, DBCS types the %DBSCED message. Generally, the "real" exception does not persist after the program-unit is recompiled with the up-to-date schema file.
01	Area not open.
02	Data base key inconsistent with area-name. Can also indicate that a referenced page number is in an area that is not in the sub-schema invoked.
03	Record affected (deleted or removed) by concurrent application.
04	Data-name invalid or inconsistent. This can occur during GET or MODIFY with a data-name list.
05	Violation of DUPLICATES NOT ALLOWED clause.

Table B-2 (Cont.)
Exception Condition Codes

Code	Condition
06	Current of set, area, or record-name not known.
07	End of set, area, or record.
08	Referenced area, record, or set-name not in sub-schema. This may occur for a number of reasons: 1. DBCS encounters a record type not in the sub-schema when traversing a set. 2. Set type owned by the object record type is not in the sub-schema. This is during a STORE or DELETE. 3. The VIA set is not in sub-schema – during set selection occurrence. 4. All subkeys are not in the sub-schema – during CALC processing or searching a sorted set. 5. The sort key of a set not in the sub-schema is modified – during a MODIFY. The solution to this is to place the required name in the sub-schema.
09	Update usage mode required. This is an attempt to use an updating verb when the specified area is open for RETRIEVAL.
10	Privacy breach attempted.
11	Physical space not available. No room remains for storing records. This can also occur while DBCS is trying to store an internal record type – namely the index blocks in a sorted set.
13	No current record of run-unit.
14	Object record is MANDATORY AUTOMATIC member in named set.
15	Object record is MANDATORY type or not member type at all in named set. This is an attempt to REMOVE a record which is either a MANDATORY member or not a member type of named set.
16	Record is already a member of named set.
17	Record has been deleted. This can occur during a FIND CURRENT of RECORD, SET, AREA, or RUN-UNIT, or during a FIND NEXT of SET or AREA.
20	Current record of run-unit not of correct record type.
22	Record not currently member of named or implied set.
23	Illegal area-name passed in area identification.
24	Temporary and permanent areas referenced in same DML verb.
25	No set occurrence satisfies argument values. This can mean, for example, that CALC value in the UWA matched no owner record.

Table B-2 (Cont.)
Exception Condition Codes

Code	Condition
26	No record satisfies rse specified. This is a catch-all exception for the FIND statement.
28	Area already open.
30	Unqualified DELETE attempted on non-empty set.
	Non-CODASYL Exception Codes
31	Unable to open the schema file.
32	Insufficient space allocated for the data-name. The SIZE clause in the data-name entry specifies less space than the compiler needs.
33	None of the areas a record type can be within are in the sub-schema.
34	A set is in the sub-schema, but its owner record type is not.
35	Dynamic use-vector is full (FORTRAN only).
36	Attempt to invoke too many sub-schemas (currently more than 8); or an attempt to use UNSET with an empty sub-schema stack or SETDB with a full sub-schema stack.
37	Sub-schema passed to SETDB is not already invoked.
38	Duplicate operation attempted on a resource. This can occur because (1) you attempt to open the journal file twice (you have opened it in EXCLUSIVE UPDATE usage-mode and are now opening a data area in UPDATE usage-mode) or (2) you call JSTRAN while a transaction is already active, or (3) you have multiple INVOKE statements and attempt to open the same area twice.
39	Data base file not found.
40	Requested access conflicts with existing access; that is, resource is not available. This can result from an attempt to 1. open an area in a USAGE-MODE incompatible with that of another run-unit using the same area (for example, trying to open an area for EXCLUSIVE RETRIEVAL while it is already open for PROTECTED UPDATE). 2. open the journal in a way that results in a USAGE-MODE conflict. 3. DELETE a record retained by another run-unit. 4. attempt to open an area or the journal and the file system signals a file-protection error.
41	No JFNs available. An attempt to open too many areas.
42	Area in undefined state (for example, after crash). DBMEND should be used to force open the area and return it to a valid state.
43	Area in creation state. This can happen to the system area only. This will occur if run-unit execution aborts at just the right time during the first OPEN of the system area. Should this occur, either rerun SCHEMA or create a 0-length file with one of the text editors.

Table B-2 (Cont.)
Exception Condition Codes

Code	Condition
44	Attempt to call a journal-processing entry point before the journaling system has been initialized (by the first OPEN that requires journaling).
45	Attempt to backup the data base with JBTRAN (1) while DBCS's Cannot-Backup-Updates (CBUU) bit is set, or (2) when the journal is shared and commands are the interleaving unit, or (3) when the journal is shared, transactions are the interleaving unit, and the argument given to JBTRAN is greater than 0.
46	Magnetic tape service is not available. DAEMDB has returned a failure code.
	System Exception Codes
55	Pseudo-exception. DBCS types message that no sub-schema yet initialized.
56	Inconsistent data in the database file. DBMEND should be used to restore the data base to a valid state. If the problem can be reproduced, it probably indicates the presence of a DBCS software error.
57	Probably a DBCS software error. If this recurs, report it.
58	Illegal argument passed by programmer or setup by host interface; for example passing a set-name with the STORE command.
59	No more memory available.
60	Unable to access a database file. The operating system reported an I/O error, either in normal operation or in trying to open a journal for appending.
61	Unable to append to journal (that is, the journal is in an aborted state but has not been designated as being done-with).
62	Attempt to enter DBCS at other than JBTRAN, SBIND, SETDB, or UNSET while the system-in-undefined-state (SUS) bit is on.
63	Unable to complete restoration of the proper data base state. This occurs either during JBTRAN or during initialization of a run-unit at the start of a command or a transaction.
64	Internal use only.
65	Monitor space for ENQUEUE entries exhausted, or ENQUEUE quota exceeded.
66	ENQUEUE/DEQUEUE failure (for example, you do not have ENQUEUE capabilities, or an unacceptable argument block has been created by DBCS).
67	Unable to initialize magnetic tape service because, for example, the IPCF block is bad; the IPCF message is too long; or DAEMDB is not running.

B.2 DBCS RUN-TIME MESSAGES

The following is a list of DBCS run-time messages. Those beginning with a left bracket ([) are for your information. If a response is required, it will be apparent to you. Those beginning with a percent sign (%) are warnings. Those beginning with a question mark (?) signal DBCS is entering an undefined state. You must then report the condition and follow procedures instituted at your facility for such occasions.

TYPE CONTINUE TO RESUME

JOURNAL CHARACTERISTICS ARE:

This is followed by a listing of the journal characteristics. Refer also to Section 2.3 for a more detailed example of this message.

%DBSCED COMPILED/EXECUTED VERSIONS OF SCHEMA DIFFER

%DBSJDM JOURNAL DEVICE MUST BE DISK OR MTA – TRY AGAIN

%DBSROA "JM" CALL REFERENCES OPEN AREA

?DBSSNI SUB-SCHEMA NOT INITIALIZED YET

?DBSUCR UNABLE TO COMPLETE RESTORATION TO PROPER DATA BASE STATE

?DBSXWX EXCEPTION WHILE PROCESSING AN EXCEPTION

B.3 COBOL COMPILER ERROR MESSAGES

The following list contains error messages from the COBOL compiler regarding DBMS syntax errors in your program. These messages can occur during compilation. Should any occur, compilation will stop.

'ALL' OR SET-NAME EXPECTED

'ALL', 'RECORD', 'AREA', 'SET', OR SET-NAME EXPECTED

AMBIGUOUS OR INCORRECT RSE SPECIFICATION

'AREA' OR 'SET' EXPECTED

AREA-NAME EXPECTED

'COMPILE' EXPECTED

'CURRENT' EXPECTED

DECLARATIVES MUST IMMEDIATELY FOLLOW PROCEDURE DIVISION

DUPLICATE SCHEMA SECTION

'ERROR-STATUS' EXPECTED

'EXCLUSIVE', 'PROTECTED', OR 'RETRIEVAL' EXPECTED

'FOR' EXPECTED

ILLEGAL COMBINATION OF ERROR-STATUS USE PROCEDURE

INCORRECT PRIVACY KEY

'INTO' EXPECTED

'INVALID', 'ONLY', 'SELECTIVE', OR 'ALL' EXPECTED

INVOKE STATEMENT MUST FOLLOW SCHEMA SECTION

NO MORE THAN 10 AREA-NAMES ALLOWED PER OPEN STATEMENT

'OF' SCHEMA OR SCHEMA NAME EXPECTED

'OR' OR 'INTO' EXPECTED

'RECORD' EXPECTED

'RECORD' OR RECORD-NAME EXPECTED

RECORD-NAME, SET-NAME, AREA-NAME, OR 'RUN-UNIT' EXPECTED

'SELECTIVE', 'ONLY', 'ALL', OR RECORD-NAME EXPECTED

'SET' EXPECTED

SET-NAME EXPECTED

SET-NAME OR 'ANY' EXPECTED

SET-NAME OR AREA-NAME EXPECTED

'STATUS' EXPECTED

'SUB-SCHEMA' OR SUB-SCHEMA NAME EXPECTED

THIS SECTION IS OUT OF ORDER

'UPDATE' EXPECTED

VARIABLE IN THIS CONTEXT MUST BE DEFINED IN SUB-SCHEMA

B.4 FORDML PREPROCESSOR ERROR MESSAGES

The following is a list of FORDML preprocessor error messages. Those beginning with a percent sign (%) are warnings. Those beginning with a question mark are fatal errors; and those beginning with a left bracket are for your information. Where appropriate, FORDML will type the line number and the line in error.

%DMLXIS.	EXTRA INPUT SPECS ARE IGNORED.
%DMLXOS.	EXTRA OUTPUT SPECS ARE IGNORED.
?DMLFSU.	SYMBOL AFTER "FIND" IS UNRECOGNIZABLE.
?DMLELW.	ENCOUNTERED ... WHILE ...
%DMLASI.	ALL MEANINGLESS SWITCHES ARE IGNORED.

?DMLWCD.	WILD CARDING IN OUTPUT DIRECTORY.
?DMLPAU.	PHRASE AFTER "FIND IDENTIFIER" UNRECOGNIZABLE.
[DMLSUM.	string, n, ERRORS AND, n, WARNINGS].
%DMLNIS.	NO INVOKE SEEN BEFORE FIRST DML STATEMENT.
?DMLOIA.	ONLY ONE INVOKE ALLOWED PER PROGRAM-UNIT.
?DMLSTL.	STATEMENT TOO LONG OR "." MISSING.
%DMLLSN.	STATEMENT NUMBER GREATER THAN 99999 – TRUNCATED.
%DMLLTL.	LINE, n, TOO LONG.
%DMLLSE.	LINE SEQUENCE NUMBER, n, NOT FOLLOWED BY "TAB".
?DMLOPF.	OPEN FAILURE FOR ",file,".
?DMLWNI.	WILD-SPEC=NON-WILD SPEC IS UNDEFINED.
%DMLCFE.	DBMS COMMENT FOLLOWED BY IMMEDIATE EOF.
%DMLESP.	EXTRA SYMBOLS AFTER "*DBMS".
%DMLICI.	ILLEGAL CHARACTER IN INPUT ON LINE, n.
?DMLSIE.	SOURCE FILE INPUT ERROR – TRY AGAIN.
?DMLCOS.	CANNOT OPEN/LOOKUP SCHEMA FILE, name,.
?DMLNSB.	NO SCHEMA BLOCK IN .SCH FILE – REBUILD IT.
?DMLBSF.	BAD SCHEMA FILE – REFERENCE IS, name, .
?DMLSSI.	SUB-SCHEMA SPECIFIED NOT IN SCHEMA.
?DMLBDK.	BAD PRIVACY KEY GIVEN.
%DMLINP.	REFERENCED NON-DATA-BASE ITEM, name, HAS NO PSEUDONYM.
?DMLDUP.	DATA BASE NAME, name, MULTIPLY DEFINED.

APPENDIX C

SCHEMA DATA DECLARATIONS: FORTRAN AND COBOL CONVERSIONS

The DBA uses the DATA ENTRY within the Schema DDL to name a data-item or data-aggregate. A date entry names and describes an alphanumeric or numeric data item, or allocates space for a data aggregate.

This appendix presents the possible Schema declarations the DBA can use and shows the FORTRAN and COBOL mappings (that is, conversions) for each. For further information, refer to Chapter 4 of the *Data Base Administrator's Procedures Manual.*

C.1 ALPHANUMERIC DATA

The USAGE phrase can be used to describe the usage mode of alphanumeric data (either data-items or data aggregates). The modes are: SIXBIT, ASCII, and EBCDIC. The corresponding schema declaration for each is: DISPLAY or DISPLAY-6; DISPLAY-7; and DISPLAY-9. Table C-1 shows the possible keyword declarations the DBA can use and the FORTRAN and COBOL conversions for each usage mode. When, for example, the schema USAGE declaration is DISPLAY-6 (PIC X(N), the FORTRAN preprocessor converts to INTEGER (N/5) for FORTRAN use. COBOL converts to DISPLAY-6 PIC X(N).

Table C-1
Alphanumeric Data: Schema Declarations;
FORTRAN and COBOL Usage-Mode Conversions

Schema Declaration	FORTRAN Usage-Mode	COBOL Usage-Mode
DISPLAY PIC X(N)	INTEGER(N/5)	DISPLAY PIC X(N)
DISPLAY-6 PIC X(N)	INTEGER(N/5)	DISPLAY-6 PIC X(N)
DISPLAY-7 PIC X(N)	INTEGER(N/5)	DISPLAY-7 PIC X(N)
DISPLAY-9 PIC X(N)	INTEGER(N/4)	DISPLAY-9 PIC X(N)

NOTE: FORTRAN rounds off to the next higher whole number if the result is not a whole number.

C.2 NUMERIC DATA

The TYPE clause can be used to describe numeric data and database keys. The types of numeric data allowed are shown in Table C-2. Table C-2 also shows the way in which each schema type declaration is treated by the host languages, FORTRAN and COBOL. If the DBA has not specified one of the numeric keywords shown in the left-hand column of Table C-2, the default is FIXED, BINARY, and REAL. The DBA can also specify the precision of each data-item. The precision is then treated as binary or decimal depending on the keyword the DBA specifies.

Table C-2
Numeric Data: Schema Declarations;
FORTRAN and COBOL Data-Type Conversions

Schema Keywords	Precision Range	Default Precision	FORTRAN Data Type	COBOL Data Type
FIXED BIN REAL	<36	35	INTEGER	COMP PIC S9 (1-10)
FIXED BIN REAL	36-70	---	INTEGER(2)	COMP PIC S9 (11-18)
FLOAT BIN REAL	<28	27	REAL	COMP-1
FLOAT BIN REAL	28-62	---	REAL*8	COMP PIC S9 (18)
FLOAT BIN COMPLEX	<28	27	COMPLEX	COMP PIC S9 (18)
FIXED DEC REAL	<19	10	INTEGER (prec/4)	COMP-3 PIC S9 (prec)

C.2.1 Schema Precision Declaration and COBOL Conversion

If you use DBMS with COBOL, you should be aware of the rules applying to legal moves and to precision when transferring numeric data within the data base. Refer to the MOVE statement specifications in the *COBOL Programmer's Reference Manual.*

The DBA can use integer-3 of the DATA ENTRY to specify precision for each numeric data-item. (Refer also to Table C-2.) Table C-3 shows decimal precision implied by each possible Schema DDL binary precision declaration. If full-word precision has not been consistently specified, left-most truncation may occur if the data-item is moved or used in computations.

Refer to Table C-3, therefore, to understand the relation between the binary precision declared in the schema and the decimal precision that results in COBOL.

Table C-3
Schema Binary Precision;
Corresponding COBOL Decimal Precision

Schema Precision Declaration (Binary)	COBOL Precision Conversion (Decimal)
1-4	PIC S9 (1)
5-7	PIC S9 (2)
8-10	PIC S9 (3)
11-14	PIC S9 (4)
15-17	PIC S9 (5)
18-20	PIC S9 (6)
21-24	PIC S9 (7)
25-27	PIC S9 (8)
28-30	PIC S9 (9)

Table C-3 (Cont.)
Schema Binary Precision;
Corresponding COBOL Decimal Precision

Schema Precision Declaration (Binary)	COBOL Precision Conversion (Decimal)
31-35 Default	PIC S9 (10)
36-38	PIC S9 (11)
39-41	PIC S9 (12)
42-44	PIC S9 (13)
45-48	PIC S9 (14)
49-51	PIC S9 (15)
52-54	PIC S9 (16)
55-58	PIC S9 (17)
59-70	PIC S9 (18)

APPENDIX D
PASSING STRING ARGUMENTS TO DBCS

This appendix is intended mainly for the FORTRAN programmer who wants to pass variable-length string arguments to the DBCS subprograms discussed in Section 2.3. Because FORTRAN does not have the facility to handle string data that COBOL has, the arguments to these DBCS subprograms are generally treated as literals (constants). To use variables, it is important to understand the relation between standard FORTRAN data types and their treatment by DBCS. Table D-1 shows this relation.

Table D-1
FORTRAN Data Types; DBCS Interpretations

FORTRAN Data Type	DBCS Interpretation
LOGICAL	data-varying
INTEGER	5 characters
REAL	5 characters
REAL*8	10 characters
COMPLEX	string pointer

The string arguments you use will be treated as data-varying strings, string pointers, or groups of characters – depending on the FORTRAN data type you specify. (Refer also to the STRLIB documentation, which discusses FORTRAN-oriented string manipulation.)

The following conventions apply for data-typing variable-length string arguments:

1. A string argument typed LOGICAL is treated as a data-varying string whose length is stored in the word preceding the string. You must provide a dimensioning statement and allocate room for the character count.

2. A string argument typed INTEGER or REAL is treated as a 5-character length string – regardless of dimensioning.

3. A string argument typed REAL*8 is treated as a 10-character length string – regardless of dimensioning.

4. A string argument typed COMPLEX is treated as a string pointer. A string pointer contains two elements: a byte pointer as its first word and the number of characters in the string as its second word. Define the byte pointer such that it points to the address of the first character of the actual string.

APPENDIX E
GLOSSARY

Area
: A named subdivision of the addressable storage space in the data base.

AUTOMATIC set membership
: A form of set membership (declared by the DBA using the Schema DDL) in which membership is established by DBMS when the record occurrence is stored.

Chain
: A method of linking records within sets. It comprises using embedded pointers within the owner and member records that make up a set occurrence.

Currency status indicators
: Single-word registers that record the data-base key of the record that is current-of-run-unit, current-of-record, current-of-set, and current of area.

Data base
: A collection of interrelated records processable by one or more applications without regard to physical storage, and defined by one schema.

Data Base Administrator
: The person or group that organizes, defines, and monitors the data base.

Data Base Control System (DBCS)
: The run-time system that acts as the interface between the run-unit and the data base.

Data-base key
: A unique identifier assigned by DBMS to each record occurrence in the data base. It remains the permanent identifier of a record occurrence until the record occurrence is deleted.

Data-item
: The smallest unit of named data in the data base.

Data Manipulation Language (DML)
: The language used by the programmer to cause data to be transferred between his program and the data base. This is not a complete language by itself; it requires a host language.

Host language
: A language into which the Data Manipulation Language has been integrated to perform actions on the data base.

Integrity of data
: The safeguarding of data from any untoward interaction of programs.

Interleaving unit
: The duration for which a run-unit retains the data base exclusively.

Location mode
: The method used for determining record storage. The location mode can be DIRECT using the unique identifier assigned by DBMS, CALC based on the CALC keys in each record, or VIA SET; i.e., according to the relationships established for the records in the set declaration.

MANDATORY set membership
: The specification of set membership (in the schema) such that once the membership of a record occurrence in a set is established, the membership is permanent. It cannot be removed from the set unless it is deleted from the data base.

MANUAL set membership
: A form of set membership in which membership is established by a run-unit by means of the INSERT command. MANUAL membership of the record occurrence in a set is declared by the Data Base Administrator when the schema is set up.

Member record
: A record, other than the owner record, that is included in a set. There may be zero or more member record occurrences in a set.

Network structure
: A general form of data structure in which any given element may be related to any other element in the structure. Networks are used to show interset relationships.

OPTIONAL set membership
: The specification of set membership such that the membership of a record occurrence in a set is not necessarily permanent.

Owner record
: The head of a group of records that make up a set. There must be one and only one record type as the owner for each set.

Privacy key
: A value that must be provided by a run-unit seeking to access or alter data protected by a privacy lock. The key must match the lock.

Privacy lock
: A value that is specified in the schema to ensure protection of the data.

Privacy of data
: The protection of data from unauthorized access.

Protected update
: A specified usage mode. It gives a run-unit the capability to make changes to an area of the data base while other run-units concurrently retrieve data.

Record
: A named collection of zero, one, or more data-items.

Record occurrence
: The actual representation of a single record. It is not the definition of a record, which is the record type.

Record-selection-expression
: The search arguments used for selecting a record from a data base.

Record type
: A specific named record defined in the DDL. It is the definition of a collection of records that have identical characteristics.

Resource
: A named entity that processes can use either shared or exclusive. A resource can be the data base, a record, a device, or a function.

Run-unit
: An executable program. A program consists of one or more program-units.

Schema
: A complete description of a data base.

Schema Data Description Language (DDL)
: The language used to describe a schema.

Sequential structure
: A data structure in which each element in the structure is related to the element preceding it and to the element following it. A form of sequential structure is used to show intraset relationships in DBMS.

Set mode
: Denotes the method of accessing the data in a set. DBMS supports CHAIN mode.

Set occurrence
: A collection of one or more logically related record occurrences. This is the actual data in the set and not its definition, which is the set type.

Set order
: The declaration of the logical order of the member record occurrences to be maintained within each set occurrence.

Set type
: A named collection of record types having one owner record type and one or more member record types.

Simultaneous-update
: The capability to update or retrieve data while another run-unit updates or retrieves data in the same area.

Sub-schema
: A description of those parts of the schema known to one or more specific programs.

Sub-schema Data Description Language (DDL)
: The language used to describe a sub-schema.

System communication locations
: Locations in core provided by DBMS for run-unit/DBCS interaction.

Temporary area
: An area not shared among concurrent run-units. A run-unit that references a temporary area is allocated a private, unique occurrence of that area. Any changes made to a temporary area are lost when the area is closed.

Tree structure
: A hierarchical structure in which each element may be related to any number of elements at any level below it, but to only one element above it in the hierarchy. Tree structures are used to show interset relationships.

User Working Area (UWA)

An area of core where all data provided by the DBCS in response to a call for data is delivered and where all data to be picked up by DBCS must be placed.

INDEX

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

Data Base Management System
Programmer's Procedures Manual
AA-0901C-TB

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or
Country

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 152
MARLBOROUGH, MA.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
200 Forest Ave.
Marlborough, MA 01752