decsystem10

DATA BASE MANAGEMENT SYSTEM (DBMS-10) ADMINISTRATOR'S PROCEDURES MANUAL

AA-0899C-TB

June 1977

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

SOFTWARE VERSION: DBMS V5
COBOL V11
FORTRAN V5

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

digital equipment corporation • maynard. massachusetts

First Printing, June 1977

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1977 by Digital Equipment Corporation

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.)

CONTENTS (Cont.)

FIGURES

CONTENTS (Cont.)

TABLES

PREFACE

This manual describes the DECsystem-10 Data Base Management System (DBMS) for the data base administrator who must create and maintain the data base. Chapter 1 briefly describes the components, concepts, terms, and functions of DBMS. Chapter 2 gives details on designing and creating the data base, while Chapters 3, 4, and 5 describe the syntax and rules for the DBMS languages. Chapter 6 contains descriptions of the DBMS utility programs. The appendices include reserved words, error messages, organization of files, and directories.

The DBMS administrator should have some knowledge of data base management systems, DECsystem-10 COBOL and FORTRAN languages, and the TOPS-10 operating system. The following manuals can provide this knowledge.

1. *CODASYL Data Base Task Group April 1971 Report*
2. *DBMS Programmer's Procedures Manual*
3. *DECsystem-10 COBOL Programmer's Reference Manual*
4. *DECsystem-10 FORTRAN Reference Manual*

CHAPTER 1
INTRODUCTION

The DECsystem-10 Data Base Management System (DBMS) is a group of programs that enable an installation to create, access, and maintain one or more data bases. DBMS is based on the 1971 CODASYL Data Base Task Group proposal.

A DBMS data base is a collection of interrelated data records structured and linked so that run-units can access them without regard to the physical storage medium. (A run-unit is the execution of a program; see Section 1.4.)

1.1 ADVANTAGES OF DBMS

DBMS data bases have two inherent advantages: removal of the data descriptions from the application programs, and centralization of data management.

Removal of the data descriptions from the application programs means that only one description of the data need exist. As a result, the programs accessing the data do not have to describe it. If the data changes, the programs do not have to change unless procedures for accessing the data change. Also, both FORTRAN and COBOL programs can access the same data because DBMS automatically gives a language-compatible form of the data to each program. In addition, you can add new data to the data base and write new programs to access the new data without changing the existing data and programs.

Centralization of the data management means that you have conceptually gathered your permanent data in one place. When the data is centralized, you can eliminate most of the duplication of the data and more easily control the integrity of your data resource. For example, you could have an employee record in the data base containing all of the necessary information for payroll, personnel, union, and government agencies.

The fact that the data is centralized does not necessarily mean that run-units can access all of the data in all of the records in the data base. You can specify the portions of the data that you will permit the run-units to access and thereby stop them from accessing the rest of the data.

A CODASYL DBMS data base has the following additional advantages.

1. It allows you to structure data in the manner most suitable to each application program although that data may be used by many programs.
2. It allows more than one run-unit to concurrently retrieve the data in the data base even while a run-unit is updating it.
3. It provides a variety of search strategies that can be used on the entire data base or portions of it.
4. It provides protection of the data base from unauthorized access as well as from destructive interaction by run-units.
5. It provides a number of ways in which you can relate the data records to each other.

1.2 COMPONENTS OF DBMS

DBMS consists of the following programs and modules.

SCHEMA –	the translator that processes the languages used to describe data bases. It also allocates and initializes the storage space for the data base. SCHEMA is described in this manual.
DBMEND –	a utility program for backup and recovery of portions of the data base. DBMEND is described in this manual.

DBINFO – a utility program that produces several reports such as cross reference listings and dumps of the data base. DBINFO is described in this manual.

FORDML – a FORTRAN preprocessor that translates data base accessing statements into FORTRAN statements. FORDML is described in the *DBMS Programmer's Procedures Manual.*

COBOL DBMS module – the module of the COBOL compiler that processes data base accessing statements. The COBOL DBMS module is described in the *DBMS Programmer's Procedures Manual.* (FORDML and this module are analogous.)

DBCS – the object-time module of DBMS used with the FORTRAN and COBOL object-time systems to access the data base. DBCS is described in the *DBMS Programmer's Procedures Manual* and referred to in this manual.

1.3 CONCEPTS OF DBMS

The terms record, set, area, and schema have a specific meaning when used with DBMS. These terms and the concepts they define are described in the following sections.

1.3.1 Records

A record is the basic retrievable unit of information in a data base. It consists of a named collection of data-items and/or data aggregates.

A data-item is the smallest unit of data in the data base. It is a subdivision of a record. For example, name, address, wage class, etc. could be data-items in an employee record.

A data-aggregate is a named collection of data-items in a record. For example, you could represent the date as a data-aggregate consisting of the data-items month, day, and year.

With traditional file processing methods, there is usually only one type of record in a file and its record description is included in the program. With data base processing, however, there can be many different types of records, so a distinction must be made between the description of the record and the actual record. In DBMS, you call the description of the record the record type and the actual record the record occurrence. Any number of occurrences of a record type can be present in a data base. Thus, if you take a file of employee records and put them into the data base, you consider the description of those records as the record type and the actual records as the occurrences.

Figure 1-1 shows the relation between a record type and record occurrences.

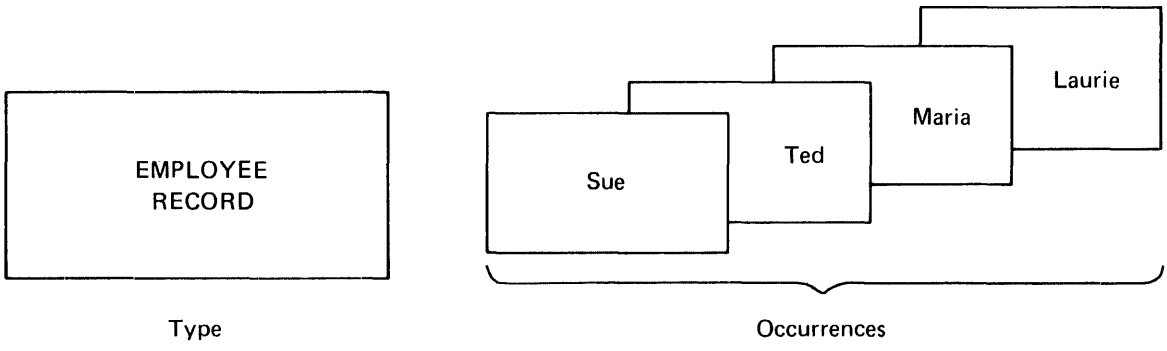


Figure 1-1 Relation Between Types and Occurrences

1.3.2 Sets

The records in a data base are not randomly arranged. They are organized into logical units called sets. A set is the mechanism by which only one physical occurrence of a record suffices for the many ways in which the programs can

access it. With an individual data file, there is no need for sets because the file itself defines the logical as well as the physical relationship among the records. However, to define another relationship, you would have to duplicate the data in another file. In a data base, on the other hand, you do not have to duplicate the records, you merely create another set. Each set defines a logical order for the records by means of pointers in the records. These pointers are described in Section 1.3.3. Because a logical relationship is described by a set, the physical location of the records on the storage device is not necessarily related to the logical order of any of the records' sets.

Like records, sets have types and occurrences. The type of a set is its description; the occurrence of a set is a group of actual records that have the relationship specified in the set description.

A set has two kinds of records in it – owner and members. A set type must have an owner record type and at least one member record type. A set occurrence does not exist until an occurrence of its owner exists. However, a set occurrence need not have any occurrences of its members. The owner of a set is somewhat like the header record on an individual file. It defines the relationship of the records in that set and is a means by which a run unit gains access to the members. For example, you could have set occurrences of employee records in which each owner is a department record and the members are employee records. The same employee records could be grouped differently in set occurrences in which each owner is a softball team record. For both sets and all other sets in which these records participate, there is only one physical occurrence of each employee record with a pointer in each for each set occurrence in which the record participates.

In DBMS, the relationship in a set occurrence is illustrated as a ring of records as shown in Figure 1-2.

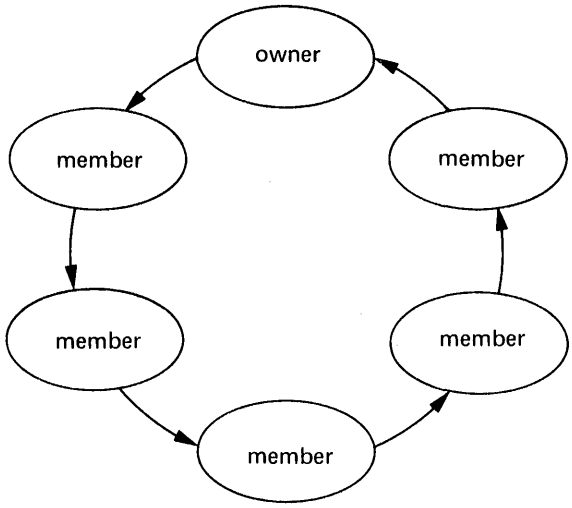


Figure 1-2 Relationship in a Set

The ring shown in Figure 1-2 has only one set of pointers; they are indicated by the arrows. Other pointers are allowed; they are described in Section 2.2.4.1.

Besides relationships among records in sets, DBMS allows relationships among sets. These relationships are the side-effect of allowing a single record type to be a member in some set types and the owner in other set types. The forms these relationships can take are illustrated in the tree structure shown in Figure 1-3 and the network structure shown in Figure 1-4.

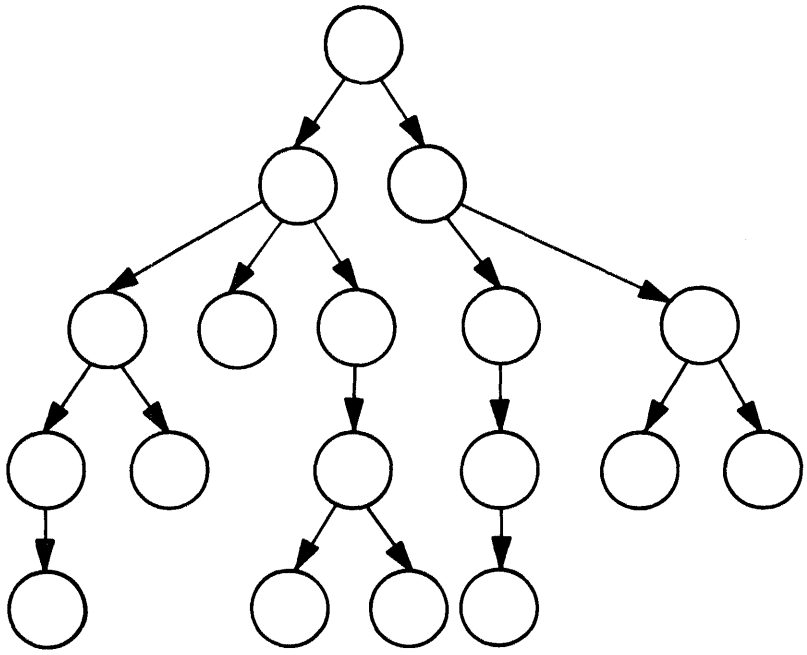


Figure 1-3 Tree Structure

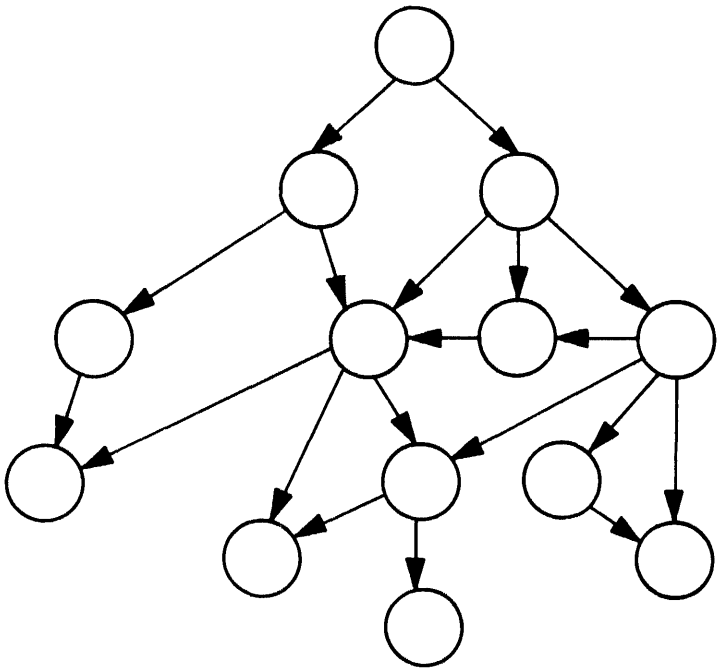


Figure 1-4 Network Structure

When a record type has no particular relationship with any other record types, you store this record type in the data base without connecting it to any sets. It can still be accessed by run-units, but not through any sets. Record types like this one must be accessed in another way. The way that records can be accessed is described in Section 1.4.1.

The following items characterize a set type:

1. A set type is a named collection of record types.
2. You can specify any number of set types.
3. You must name each set and you must specify one owner record type and one or more member record types for it.
4. You must specify a SET ORDER for each set. (See Section 2.2.4.2.)
5. You can specify any record type as the owner record type of one or more sets.
6. You can specify any record type as a member record type of one or more sets.
7. You can specify any record as both an owner record type in one or more set types and a member record type in one or more different set types. However, DBMS does not allow a record type to participate as both owner and member in the same set type.

The following items characterize a set occurrence:

1. A record occurrence cannot appear in more than one occurrence of the same set.
2. The existence of the owner record occurrence distinguishes that set occurrence from all other occurrences of that set type.
3. In addition to the occurrence of its owner record, a set occurrence can have any number of occurrences of each of its member record types.
4. A set occurrence that contains only an occurrence of its owner record is known as an empty set occurrence.
5. You can specify a set type in which the owner is SYSTEM. There is only one occurrence of this set type and it is called a singular set.

1.3.3 Areas

An area is a named subdivision of the storage space in a data base. The occurrences of records are stored in areas.

DBMS provides the ability to subdivide the data base so that you can segregate data according to its use. Then, each run-unit need only access selected areas rather than the entire data base. In addition, you can perform backup selectively according to the frequency of use of the data. For example, an area containing data for retrieval only (e.g., tax tables) need not be backed up as often as an area containing data that is frequently updated.

Each area is divided into fixed-length physical units called pages. You specify the size of a page when you define each area (see Section 2.2.2.2). A page logically consists of some number of lines. Each line is a group of 36-bit words large enough to contain a single record occurrence and its set pointers. The size of a record occurrence, and therefore of the line containing it, depends on the data-items and data-aggregates that you specify it to contain and the number of set pointers in it. All occurrences of the same record type are the same size, but size varies among record types.

Every page in an area is numbered consecutively starting and ending with the numbers you specify in the DMCL area entry (see Chapter 3). A page's page number is stored in the header at the beginning of the page. (See Appendix D for the format of the header.) Each line on each page is numbered consecutively starting with 1; and lines cannot cross pages. There are as many lines on a page as will fit into the page size you specified.

Because each record occurrence is one line, the combination of the page number and the line number for any record occurrence is its address (i.e., uniquely identifies it). This address is called the database key. Every record occurrence stored in the data base is assigned a database key and thus can be located by means of its database key. Consequently, database keys are used as the set pointers in a record occurrence. That is, each record occurrence contains the database key of at least one other record occurrence for each set in which the record occurrence is an owner or member. The SCHEMA program automatically creates the space for the set pointers in each line and DBCS auto-

matically puts the database keys into that space when a run-unit manipulates a record occurrence. You need only specify the sets in which the records participate.

1.3.4 Schemas and Sub-Schemas

A schema is a description of a data base. It contains definitions of all the areas, set types, and record types (including data-items and data-aggregates) in the data base. A schema does not contain the data, only the description of the data. You must describe each data base with a separate schema.

A sub-schema is a subset of a schema and contains descriptions of those areas, set types, record types, data-items, and data-aggregates known to one or more run-units. You can define up to 36 sub-schemas in conjunction with each schema. That is, the sub-schema can only contain areas, set types, and record types that are in the schema. You define a sub-schema to restrict a run-unit's access to only those portions of the data base that it needs. Each sub-schema that you define can contain the same or different portions of the schema as another sub-schema. That is, the information in sub-schema can overlap.

In a sub-schema, you can include a record type and omit some or all of its data-items or data aggregates. However, you cannot include a data-item or data-aggregate without including its record type. Also, you can define the components of a data-aggregate differently in each sub-schema. This means that you can specify the data-aggregate in COBOL terms for COBOL programs and in FORTRAN terms for FORTRAN programs.

The statements used to define the schema are divided into three languages, but are processed by one translator (the SCHEMA program). The three languages are:

1. Device Media Control Language (DMCL)
2. Schema Data Description Language (DDL)
3. Sub-schema Data Description Language (DDL)

You use the DMCL to describe the overall characteristics of the schema and the physical characteristics of the areas in the schema. For example, you can name the journal used with that schema (see Section 1.4.3) and the size of an area's pages. Use of the DMCL is described in Chapter 2 and its syntax and rules are given in Chapter 3.

You use the DDLs to describe the schema and the sub-schemas. The use of both DDLs is described in Chapter 2. The syntax and rules for the schema DDL are given in Chapter 4, for the sub-schema DDL in Chapter 5.

1.4 ACCESSING THE DATA BASE

Once you have designed your data base, you must design the applications that will use it. Each application can be one or more programs written in COBOL or FORTRAN. These programs will become the run-units that access the data in the data base.

A run-unit is the execution of a program. That is, it is the compiled, linked, and running program. The distinction between a run-unit and a program is made in this manual in keeping with the CODASYL Data Base Task Group's distinction.

1.4.1 Data Base Accessing Language

When the programmers write the data base application programs, they must include statements of the DBMS language that accesses the data base. This language is the Data Manipulation Language (DML).

The DML is not a complete language by itself, but a host-language extension. In other words, the DML relies on the host language (COBOL or FORTRAN) to offer the framework from which the DML can provide the interface with the data base. In an application program, the DML statements and the host language statements coexist freely, and the distinction between them is merely conceptual. From the programmer's point of view, he is using one, unified language that has the capabilities of both the host language and the DML. Chapter 3 of the *DBMS Programmer's Procedures Manual* contains the syntax and rules for the DML.

The DML provides the statements that access the data in the data base. With these statements, the run-unit can:

1. Specify the sub-schema it will use (INVOKE).
2. Open and close areas (OPEN and CLOSE).
3. Find a record occurrence in the data base (FIND). The run-unit can use one of several search strategies (called record-selection-expressions) to find the record occurrence:

   a. by means of any set occurrence in which the record occurrence participates as long as the set occurrence is in the sub-schema.
   b. by means of a search of an area.
   c. by means of the record occurrence's database key.
   d. by means of data-items in the record occurrence itself.

4. Get the record occurrence from the data base (GET).
5. Store a record occurrence in the data base and insert it into one or more sets (STORE).
6. Insert a previously stored record occurrence into one or more sets (INSERT).
7. Delete a record occurrence from the data base and thus remove it from all set occurrences (DELETE).
8. Remove a record occurrence from one or more set occurrences, but not delete it from the data base (REMOVE).

In addition, the run-unit can modify data, save the contents of special registers, test for errors or other conditions, and perform error recovery.

The COBOL compiler contains a DML module that directly processes the DML statements. Thus, the COBOL programmer can include DML statements in his program and compile it like any other COBOL program. The FORTRAN programmer, however, must pass his program through a preprocessor called FORDML before he compiles it. This is because the FORTRAN compiler does not contain a DML module.

1.4.2 Data Base/Run-unit Interaction

Any run-unit must work through an object-time system to communicate with the TOPS-10 operating system. A data-base accessing run-unit must then work through a DBMS object-time system. This object-time system is called the Data Base Control System (DBCS). It is the interface between the run-unit and the data base; i.e., DBCS is the module that actually performs the actions defined by the DML statements.

When the run-unit invokes a sub-schema, DBCS creates a User Working Area (UWA) for the program. The UWA is a loading and unloading zone where all data provided by DBCS in response to a call for data is delivered and where all data to be picked up by DBCS must be placed. Each program or subprogram has access to the UWA, which contains a declaration (i.e., local copy) of each record in the invoked sub-schema. The data in the UWA is not disturbed except in response to the execution of a DML command or by the run-unit's host language statements. UWA locations are not necessarily contiguous.

The UWA is set up by DBCS in accordance with the invoked sub-schema. Each data-item included in the sub-schema is assigned a location in the UWA and may be referenced by its name as declared in the sub-schema. Data items included in the data base, but not in the sub-schema invoked, are not in the UWA and cannot be referenced.

DBCS also provides for a number of System Communication Locations. These locations are used for run-unit/DBCS interaction and are part of the UWA. There are locations for DBCS to monitor the names of the current area and record; the error status; the names of the set, record, and area in which the error occurred; and the error count.

The diagrams in Figures 1-5 and 1-6 illustrate the program building processes from source to run-unit.

1.4.3 Backup/Recovery During a Run-Unit

When an updating run-unit accesses the data base, it has the potential of making damaging changes to the data there; so DBMS provides a means for backing up the changes made during a run-unit. It involves use of a journal file. Each time the run-unit updates a page in the data base, DBCS writes a BEFORE image of that page in the journal. A BEFORE image is a copy of a data base page before the page is changed. You must make provisions to use this feature; see Section 2.2.7. Also see Chapter 6 for a description of DBMEND, the recovery utility program.

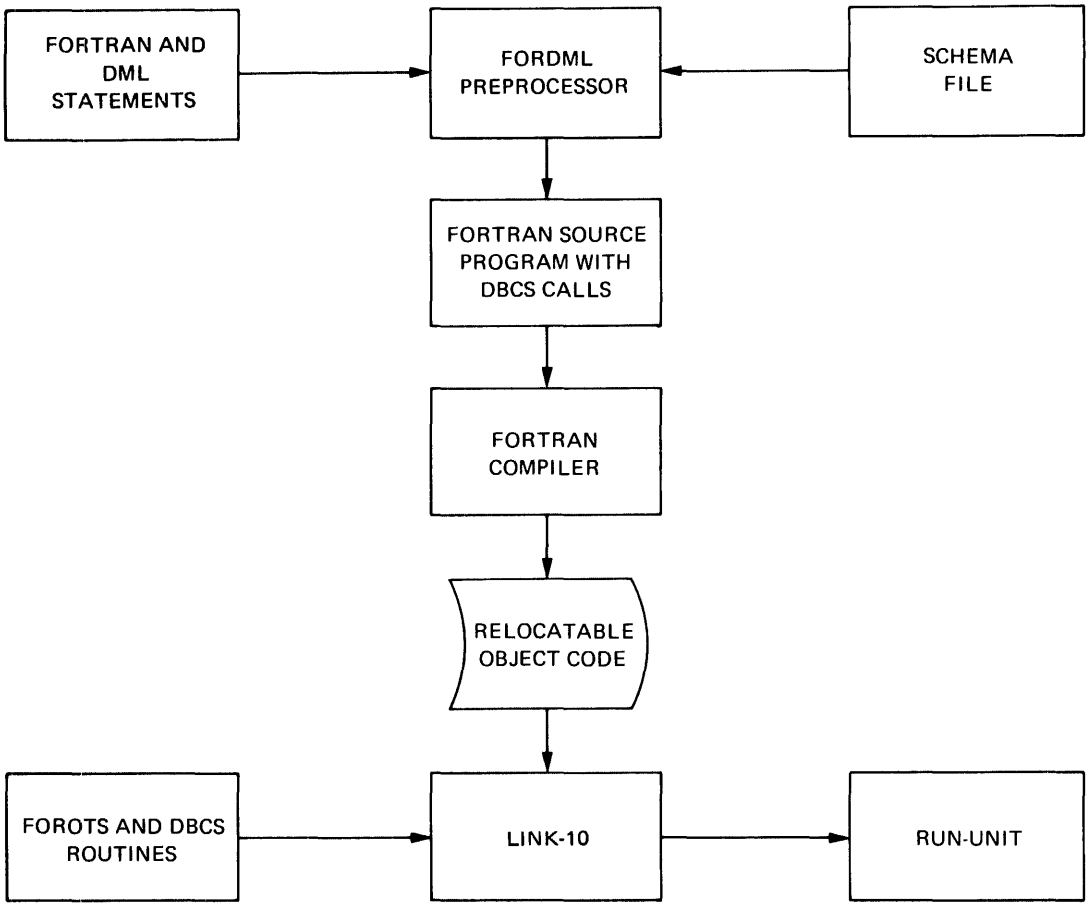


Figure 1-5 Program Building Process for FORTRAN Programs

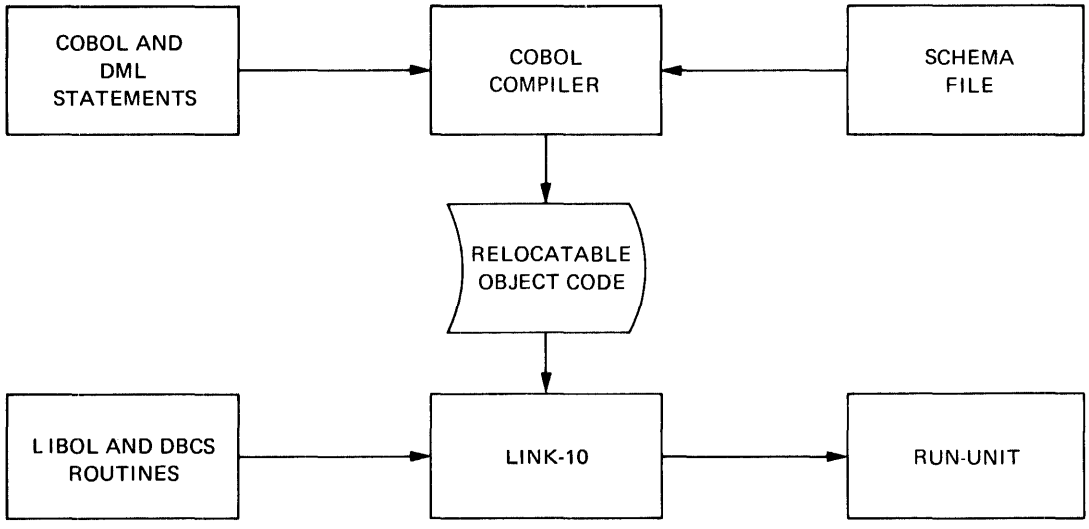


Figure 1-6 Program Building Process for COBOL Programs

You can also have AFTER images in the journal file. They are copies of data base pages after the pages have been changed. You would use the DBMEND program to process the AFTER images if you wished to bring an old copy of a data base up to date.

The run-unit can also specify characteristics of and cause information to be placed in the journal file. See Section 2.3 in the *DBMS Programmer's Procedures Manual.*

1.4.4 Exception Handling During a Run-Unit

In addition to backup/recovery, DBMS also provides for exception handling during a run-unit. The mechanism of exception handling is that DBCS notifies the run-unit if an error (called an exception) occurs. DBCS signals an exception to a run-unit by placing an error status code in one of the special registers. The run-unit can then test this register after each DML statement in case an exception occurred. See Appendix B in the *DBMS Programmer's Procedures Manual* for a description of the error codes.

Also, you can have DBCS notify the programmer of any or all exceptions and then wait for the programmer to decide to continue the run-unit or stop it, depending on the nature of the exception. See Section 2.2.8 for more information about specifying exception interception.

CHAPTER 2

RESPONSIBILITIES OF THE DATA BASE ADMINISTRATOR

As the Data Base Administrator, you have the responsibility to control the data base(s) at your installation. Some of the duties that this entails are described in this chapter. They include designing the data base, creating and maintaining the schema, and using the DBMS utilities to protect and document the data base. Additional functions may be required at your installation such as training and personnel management. Since such functions are installation-dependent, they are not discussed in this manual.

2.1 DESIGNING THE DATA BASE

The first duty you must perform is design of the data base(s) according to the applications that are currently in use or planned for your installation. You should examine the data and the usage of that data to determine its relationships. Depending on the usages and relationships among the data, you may decide to establish one or more data bases.

Once you have decided on the relationships among the data, you should define the actual set relationships that exist among the record types and define the owners and members in the sets. You can use the tree or network structures to show the set relationships and you automatically use the ring structure to show the relationships of members in each set.

You must also define the application programs that will access the data base. If programs exist that access the data in its current form, these programs will have to be revised to interact properly with the data base.

You will have to maintain close communication with the programmers writing the application programs. For example, you will want to give them copies of the schema and sub-schemas that their programs will use. Once they have examined the schema and sub-schemas, you should explain the way in which their programs will use the information in the schema and sub-schemas.

You should also consider two points about future growth of the data base:

1. Allocating enough space
2. Adding new record and set types

You should allocate enough space in the data base for future growth of existing applications and for new applications. You can do so by specifying a page range for your areas larger than you currently need. See Section 2.2.2.2 for detailed information about specifying page ranges.

If you know that you will need new record and set types and know what they will look like, you should put them in the schema when you create it. Their existence in the schema does not mean that you have to include them in any sub-schemas until you need them. This is easier than adding new record and set types to an existing schema; see Section 2.4 for more information.

2.2 CREATING THE DATA BASE

Once you have designed your data base, you must create it. To do so, you will perform the following steps:

1. Write the schema using a text editor.
2. Process this file using the SCHEMA program.
3. Load data into the resulting data base.

To write the input schema file, you must include the DMCL and DDL statements that describe your data base. You can use a system editor to create this file. The contents of the schema file are described in this chapter. The DMCL and DDL statements are described in Chapters 3, 4, and 5.

To process the file, you run the SCHEMA program with the source schema file as input. SCHEMA produces an object schema file and one zero-length file for each area in the data base. Section 2.3 describes how to process the schema file.

To load the data base, you must write one or more application programs that store data in the data base. These programs can be your installation's application programs or special ones that are used only for this purpose. Because loading is specific to each installation's data base, it is not described further in this manual.

When planning the schema file, you should consider the following points:

1. How to describe areas
2. How to describe records
3. How to describe sets
4. How to describe sub-schemas
5. How to describe the journal
6. How to describe exception handling.

These points are covered in detail in the following sections. They are not necessarily ordered in the way in which you must write the schema. The order that must be maintained in the schema is:

1. DMCL entries
2. Schema DDL entries
3. Sub-schema DDL entries

2.2.1 Describing the Logical Attributes of Areas

You define the logical attributes of an area in the Schema DDL Area Entry. You can specify the name of the area, the privacy locks for each usage-mode, and whether or not the area is temporary.

2.2.1.1 Naming Areas in the Schema DDL – An area-name can be up to 30 characters long. The name you give in the DDL must match one of the area-names in the DMCL. You must specify at least one area for the data base. If you specify only one, that area is, in effect, the entire data base.

2.2.1.2 Specifying Privacy Locks for Areas – An area can be opened by a run-unit in one of six usage-modes. You can specify a privacy lock for each usage-mode, one for all usage-modes, or none. The usage-modes are as follows:

1. EXCLUSIVE RETRIEVAL – only that run-unit can access the area, but cannot update it.
2. EXCLUSIVE UPDATE – only that run-unit can access the area, and can update it.
3. PROTECTED RETRIEVAL – all run-units can access the area, but none can update it.
4. PROTECTED UPDATE – all run-units can access the area, but only that run-unit can update it.
5. RETRIEVAL – all run-units can access the area and run-units open for PROTECTED UPDATE can update it.
6. UPDATE – all run-units can access and update the area.

The privacy locks are a means of protection for the areas of the data base. The application program must give a privacy key that will match the lock before it can open the area. The key has the same value as the lock, and DBCS compares them before opening the area. As long as they remain known only to those using the data base, these locks and keys can help to ensure the privacy of the data base. You will have to give the values of the privacy keys to the programmers writing the applications so they can use the correct ones in their programs.

2.2.1.3 Specifying an Area as Schema Temporary – When you specify an area as TEMPORARY, you are specifying an area that does not have permanent data in it. Each run-unit receives its own copy of the area and must load data into its copy. When the run-unit closes the area, DBCS discards the data and any changes made to it. Thus, a temporary area is a useful debugging tool because new run-units can access the data base without disturbing the actual data in it.

2.2.2 Describing the Physical Attributes of Areas

The DMCL Area Entry allows you to describe the physical attributes of each area in the data base. You can describe an area's file specification, its size, the size of its pages, its number of buffers, its number of CALC-chains per page (see Section 2.2.3.2), and the page range of individual record types in it.

2.2.2.1 Specifying an Area's File Specification – When you create an area, you are actually assigning that area to a file; you must, therefore, give the file specification for the area. The specification can contain up to six characters; you cannot specify the extension. SCHEMA automatically assigns the extension .DBS to the file. A DBS file is thus one area of the data base. When you create files in this way, you must give them unique names under the project and programmer numbers to which you have assigned the data base. You can, in turn, create the data base under any number of project and programmer numbers known to the monitor.

Note then that the programmers running the application programs must be able to access the project and programmer number you have assigned the data base. You can ensure this in three ways.

1. Assign the programmers to the same project programmer number as the data base. Set the protection code for the data base to 077 so that no other users can access it.
2. Assign the programmers to the same project number as the data base, but to different programmer numbers. Set the protection code for the data base to 027. This allows all programmers with the same project number as the data base to access it, but no others to access it. You should then request that the project number you assigned to the data base be reserved for data base use only; this prevents non-data base users from accessing the data base. Note that to allow a programmer to create a file in the data base directory, you must set the protection code for the directory to 770.
3. Assign the programmers to project programmer numbers different from those you assign to the data base. Set the protection code for the data base to 022 so that all project programmer numbers can access the data base. Then set the User File Directory protection code to 777 so that all project-programmer numbers can create files in the directory of the data base. Be aware, however, that users other than your application programmers can access – and possibly alter – the data base.

When you assign the area to a file, you can specify that the area will be used as the SYSTEM area; that is, the SYSTEM record will be stored in this area along with the other records you designate. The SYSTEM record is declared in the OWNER clause in the schema DDL. If you do not explicitly designate a SYSTEM area but do specify a SYSTEM record, DBCS will store the SYSTEM record in the first area that you assigned.

2.2.2.2 Specifying Sizes for an Area – In the DMCL you must specify the size of the area in terms of its range of pages. That is, you give the number of its first page (FIRST PAGE clause) and that of its last page (LAST PAGE clause). When you have more than one area in a data base, you cannot overlap the page ranges, but neither do you have to number them consecutively. Thus, each area has its page range within an overall page range for the data base. By defining each area's range, you implicitly define the range of the data base. For example, if you have three areas with ranges 1 to 100, 125 to 250, and 300 to 500, the range of the data base is 1 to 500. Note that non-contiguous page ranges will provide you with a mechanism for expanding the size of an area if the need arises. However, read Section 2.2.2.5 carefully for more information about expanding an area's page range.

You must also specify the page size and a limit to the number of records that can be stored on a page in the area. You give the page size in terms of 36-bit words and SCHEMA automatically rounds up to the next multiple of 128 words to match the monitor block size. The limit of records per page means the limit of lines on a page. You can specify this number for all the areas, or for each one separately. You must balance the number of records per page against the number of words you specify for a page. If you specify too small a number of records per page, you may have extra space on the page after the maximum number of records is reached.

When you are designing your data base, you should plan for future growth. One possible way is by making the page size and/or possibly the page range larger than you currently need. To calculate the amount of space you want, add the amount of space you currently need and the rate of growth over the time you wish this data base to exist without reloading. Remember the page size you specify is actually the logical page size. DBCS will not place more than this number of words of data on a data-base page. Having a logical page size that is less than the physical page size allows you to distribute the data across pages such that you can allow for future growth.

2.2.2.3 Specifying the Number of Buffers for an Area – You can also specify the number of buffers that DBCS will use for the area. The size of the buffer is set equal to the size of a page, so you may want to consider the amount of memory required for each buffer before you calculate the number of buffers. If you do not specify the number of buffers, DBCS will use three.

DBCS allocates the buffers when a run-unit opens the area. It returns the buffer space to free storage when the run-unit closes the area.

2.2.2.4 Specifying the Number of CALC-Chains per Page for an Area – You can specify the number of CALC-chains for each page of an area. If you do not, the SCHEMA program will assign one CALC-chain to each page. A CALC-chain is a group of pointers, the first of which is a word in the page header. This word points to the first CALC record in the chain, and that record has a pointer to the second record in the chain and so on. (For more information about CALC records see the discussion of CALC LOCATION MODE in Section 2.2.3.2.) To find a record by means of its CALC key, DBCS must perform a linear search of the chain for the CALC record by starting with the CALC-chain header and following the chain from one CALC-record to the next until it finds the desired CALC record. Figure 2-1 shows an example of this.

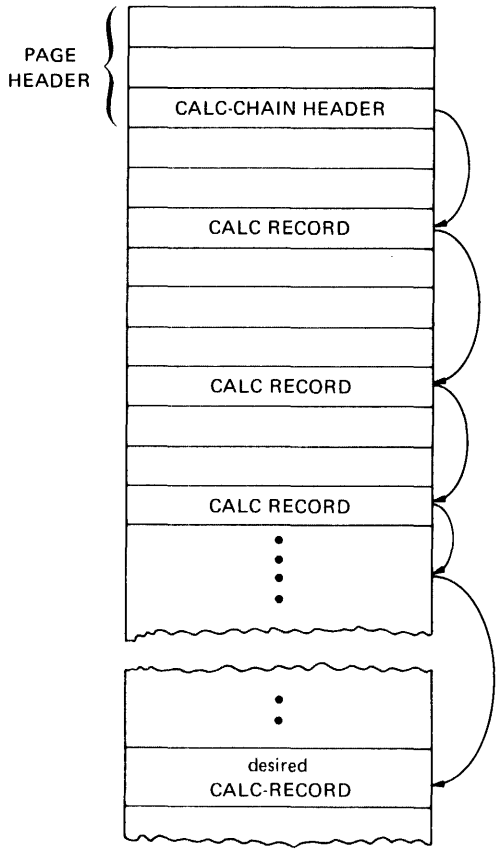


Figure 2-1 Following a Single CALC-chain

However, when you increase the number of CALC-chains per page, you increase the number of initial pointers in the page header so that the chains through the CALC records tend to be shorter than if only one CALC-chain existed.

DBCS would then have to search through fewer CALC records to find the desired record. Thus, you can trade off fast CALC-record access and space overhead per page. Use of multiple CALC-chains (2) is shown in Figure 2-2.

If you do not plan to have any CALC records in an area, you can specify no CALC-chain headers and save the space on each page of the area. Note that you should not specify so many CALC-chains per page that many will never be used. For example, if you have 100 pages and will never have more than 200 CALC records in the area, (100*n CALC-chains) should not be greater than 200.

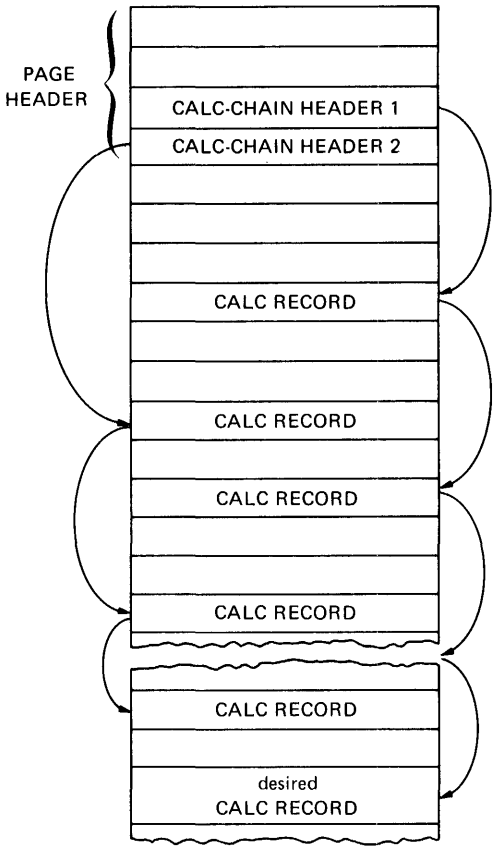


Figure 2-2 Following Several CALC-chains

2.2.2.5 Specifying Ranges for Records in an Area – Within each area, you can specify a page range for each record type that can be stored in the area. The range specifies the pages on which occurrences of the record type can be stored. If you do not specify a page range for a record type, its range is the entire area. This capability is useful for planning for CALC records and for clustering occurrences of records.

When you plan for CALC records, you should specify a page range for each CALC record type. If you thus limit the page range for each CALC record type, you can increase the last page of the area without losing the ability to access the previously-stored occurrences of the CALC records. This is because the CALC algorithm uses the number of pages in the CALC record's range to store and retrieve the record occurrences. If you do not specify a page range for the CALC record, its page range would be the entire area. If you then increased the last page of the area, DBCS could not locate the previously-stored record occurrences.

When you specify the page range for a CALC record (either implicitly or explicitly), you should make the number of pages an odd number. Because of the way the CALC algorithm works, the odd number will tend to cause a more even dispersion of the CALC record occurrences throughout the area. (You will always get an odd number of pages if you make the first page and last page of the range both odd numbers or both even numbers.)

You can also use the page range to cluster the occurrences of a record type into a chosen sequence of pages.

2.2.3 Describing Records

In the Schema Record Entry in the DDL, you define each of the record types that can reside in the data base. You name the record and give its location mode, the areas in which it can reside, and its data-items and data-aggregates.

2.2.3.1 Specifying the Name of a Record – For each record type in your data base, you must give a name up to 30 characters long. When you name the record type, you also implicitly give it a record type ID. The SCHEMA program actually creates this internal, numeric record type ID for each record type that you specify. It assigns the record type IDs in ascending numeric order as you specify the record types. For example, the first record type you define gets the first available record type ID; the second record type, the second record type ID; and so on. (The system uses the beginning record type IDs for the system record types.) The record type ID is a permanent identifier of the record type and cannot be altered unless you completely recreate and reload the data base. Consequently, you cannot add a record type among those that already exist; you must place new record types after the last existing record type (see Section 2.4).

2.2.3.2 Specifying the Location Mode for a Record – The LOCATION MODE clause allows you to specify how a record is to be stored in the data base. You can specify one of three location modes for a record:

1. DIRECT – the record occurrence will be stored according to the database key in its DIRECT key. You must provide an identifier for the DIRECT key because the database key is not a part of the record.
2. CALC – the record occurrence will be stored according to data-items in the record. You must identify these data-items. You must also specify if these data-items can have the same value in different records.
3. VIA – the record occurrence is stored according to one of the sets in which the record resides.

DBCS uses a different algorithm for each location mode to try to find the appropriate page on which to store the record occurrence. If there is no space left on the page it finds, DBCS uses another algorithm to find a page on which to store the record occurrence. (See Appendix D.)

When you specify DIRECT, DBCS uses the value in the DIRECT identifier, which can be a database key supplied by the run-unit or zero. If the value is not zero, DBCS uses the page number portion of the database 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.

When you specify CALC, DBCS hashes the values in the data-items you specified as the CALC keys and uses the result to locate the appropriate page and the CALC-chain on the page.

When you specify VIA, DBCS locates the page of the member that is logically prior to the new record in the VIA set. If the logical prior record is not in an area that the new record can be within, DBCS uses the page of the current record of the area for the new record. If there is no current record of the area or if the current record of the area is not in the new record's page range, DBCS uses the first page of the range for the new record.

You should declare the location mode as VIA for each member type that is usually accessed serially (i.e., when the application program usually specifies FIND NEXT). This will physically localize each set occurrence and reduce page accesses by DBCS. You should only declare records as CALC when they must be accessed randomly. Otherwise, there is no justification for the overhead involved in maintaining the CALC-chains.

You can best create multilevel structures that span areas if you specify the location mode as VIA for the member records in the sets in the structures. An example of a multilevel structure is shown in Figure 2-3.

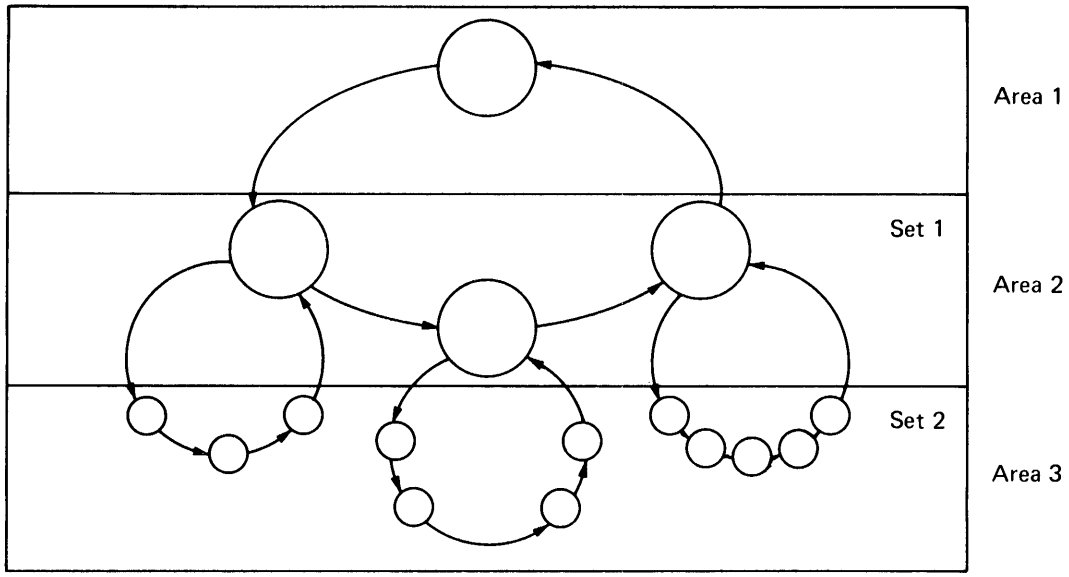


Figure 2-3 Example of a Multilevel Structure

Because each area has its own set of buffers, the run-unit accessing the sets in the structure has many windows into the data. That is, DBCS can go serially through the members of one set in one area, without losing its place in another set in another area.

2.2.3.3 Specifying Areas for Records – You must declare the areas in which occurrences of a record type can be stored. If you specify more than one area, you must also specify an identifier that is known as the area-ID. The run-unit uses this area-ID to tell DBCS the particular area where the record occurrence will be stored (or, in some cases, accessed).

2.2.3.4 Specifying Data-Items in Records – You specify the structure of a record by naming the data-fields that make up the record. You can specify the data as elementary numeric or alphanumeric items or as data-aggregates. You use the PICTURE clause to describe elementary alphanumeric data-items, the SIZE clause to describe data-aggregates, and the TYPE clause to describe elementary numeric items. The exact structure of these clauses and their meanings can be found in Chapter 4. While numeric and alphanumeric data-items cannot be further subdivided, data-aggregates can. However, you can describe the components of a data-aggregate only in the sub-schema DDL. This allows you to change the description of the data in the data-aggregate according to the use that will be made of it.

2.2.4 Describing Sets

Once you have defined the records in the data base, you then define the relationships among these records by means of the Set Entry in the schema. With the Set Entry you can define the mode, order, and owner of the set.

2.2.4.1 Specifying the Mode of a Set – The current implementation of DBMS provides for only one set mode – CHAIN. For each occurrence of a set, a chain of pointers is created that provides for serial access to all records in the set occurrence. The pointers are embedded in the records themselves. An illustration of an embedded pointer chain is shown in Figure 2-4. It represents a set occurrence with two member records. The owner record of the set occurrence contains a pointer to the first member record in the set which in turn contains a pointer to the second

member which points back to the owner. If the set occurrence contained n member records, the chain of pointers would pass through the n member records.

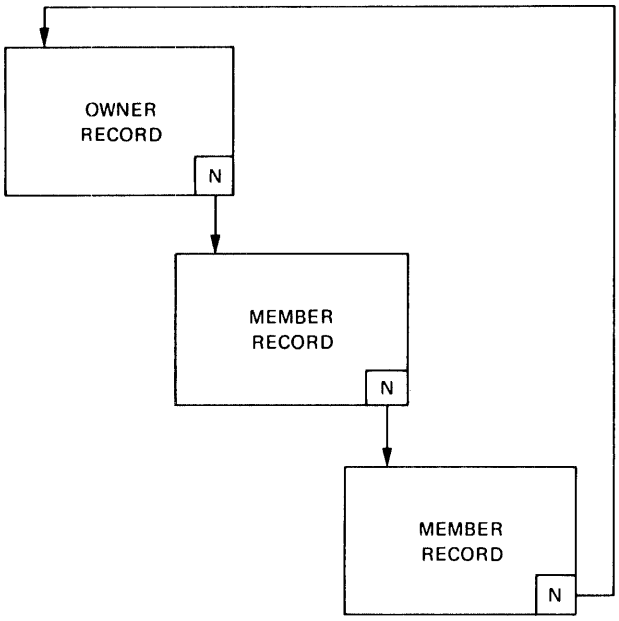


Figure 2-4 Chain with NEXT Pointers

The linkage provided between the records in a chain is only in the NEXT direction unless you specify the optional clause LINKED TO PRIOR. When you specify LINKED TO PRIOR, links in the reverse (i.e., PRIOR) direction are also present. Figure 2-5 is a representation of an embedded chain that is LINKED TO PRIOR.

In addition, you can specify that the occurrences of any of the member record types specified for a set are LINKED TO OWNER. This causes the owner record of the set occurrence to be accessible directly from each of the member record occurrences. Figure 2-6 illustrates this.

The database keys are used as pointers. DBCS assigns space for a minimum of one pointer (the NEXT pointer) for each record occurrence for each set occurrence in which the record occurrence participates as owner or member. DBCS assigns additional pointers and space if you specify the chain to be LINKED TO PRIOR or you specify the set's members to be LINKED TO OWNER.

When records in a set are frequently updated (particularly when they are being deleted), PRIOR pointers in those records in addition to the always-present NEXT pointers will improve run-unit efficiency.

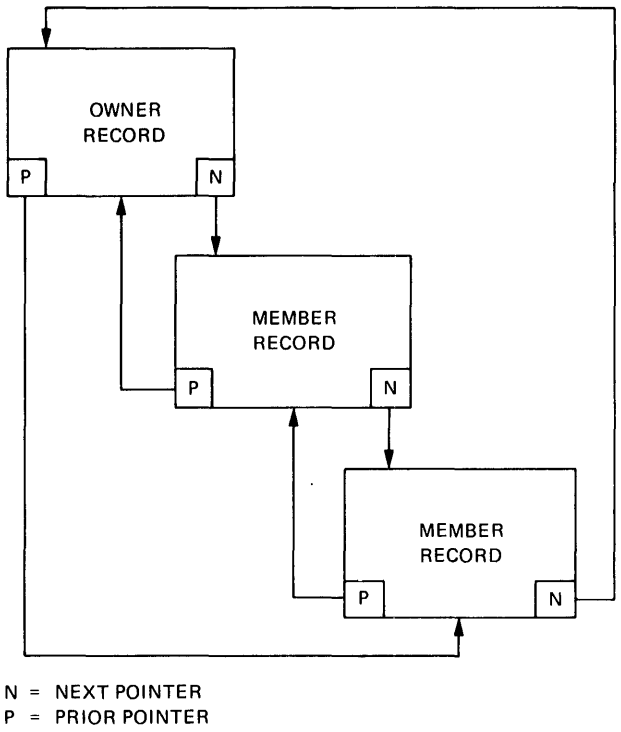


Figure 2-5 Chain with NEXT and PRIOR Pointers

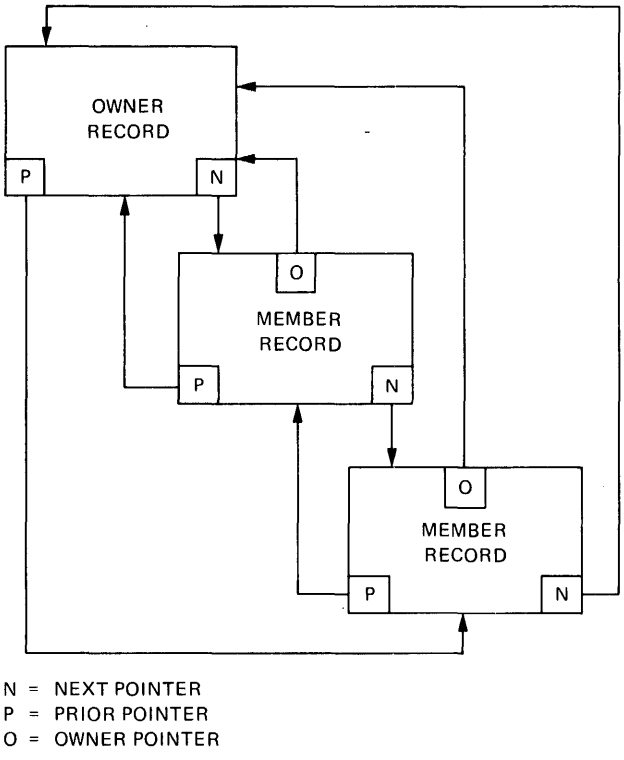


Figure 2-6 Chain with NEXT, PRIOR and OWNER Pointers

2.2.4.2 Specifying the Order of a Set – You must specify a SET ORDER clause for each set you name in the schema. The SET ORDER controls the logical order of the member record occurrences within each set occurrence. The logical order of the member records of a set is completely independent of the physical placement of the records themselves. Thus, the same record occurrences could participate as members in occurrences of two different set types and be ordered differently in each of those set types.

The member records of each occurrence of a given set may be ordered in one of two 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 member records' type IDs, or their database keys, or any combination of these.
2. In the order resulting from inserting new member record occurrences into the set:

   a. FIRST, that is, as the immediate successor to the owner record occurrence.
   b. LAST, that is, as the immediate predecessor to the owner record occurrence.
   c. NEXT, PRIOR, that is, after or before the current record of the set.

Unless sorted order is necessary (e.g., for sorted reports), the overhead that it involves is not justified.

2.2.4.3 Specifying the Owner of a Set – With the OWNER clause, you can specify the name of a record that will be the owner of the set or that the owner is SYSTEM. If you specify SYSTEM, you create an inherently singular set, i.e., a set that has only one occurrence. The SYSTEM record is stored in the area that you specified for it in the DMCL or by default in the first area you declared (if you did not explicitly specify a SYSTEM area).

2.2.5 Describing the Members of a Set

You use the MEMBER entry to specify the names and characteristics of the member records in the set. You must completely define each member before you define another member. The characteristics of a member are the type of membership, its keys, and the method of set occurrence selection.

2.2.5.1 Set Membership – You can specify the membership of a record type in a set as AUTOMATIC or MANUAL, and MANDATORY or OPTIONAL. A record type can have different forms of membership in different sets.

AUTOMATIC/MANUAL refers to how a run-unit puts record occurrences into the data base and into set occurrences. AUTOMATIC means that a run-unit uses a STORE statement to automatically insert a record occurrence into the set occurrences in which it is an automatic member as well as to store the record occurrence in the data base. MANUAL means that a run-unit must use an INSERT statement to manually insert a record occurrence into the set occurrences in which it is a manual member. A run-unit must have previously stored a record occurrence in the data base before that record occurrence can be manually inserted into a set occurrence. For example, if you have an employee record type, you could give it automatic membership in the company set type and manual membership in the department and stock-option set types. When an occurrence of the employee record is stored, it will automatically become a member of the company set occurrence. It becomes a member of the department and stock-option set occurrences only when a run-unit inserts it into those set occurrences.

MANDATORY/OPTIONAL refers to how permanent a member record occurrence is in a set occurrence. MANDATORY means that membership in a set is permanent. A run-unit cannot use a REMOVE statement to remove a record occurrence from any set occurrence in which it is a mandatory member. However, the run-unit can use a DELETE statement to eliminate the record occurrence from the data base and thus from all set occurrences in which it is a member. The run-unit can also eliminate a mandatory member from the data base by deleting its owner record. OPTIONAL means that membership in a set is not permanent. The run-unit can use a REMOVE statement to remove a record occurrence from any set occurrence in which it is an optional member. The run-unit can use a DELETE statement to eliminate the record occurrence from the data base and thus from all set occurrences in which it is a member. The run-unit can also eliminate an optional member from the data base by deleting its owner record using a special form of the DELETE command. For example, with an occurrence of the employee record mentioned above, a run-unit can remove it from an occurrence of the stock-option set as long as it is an optional member of that set. However, the employee record should be a mandatory member of the company set so that a run-unit can remove an occurrence of that record from that set only when it deletes that occurrence from the data base.

2.2.5.2 Sort Keys – You can specify the sort keys for a member of a sorted set by means of the ASCENDING/ DESCENDING phrase. You can also specify that the keys are RANGE KEYs. This means that the keys will be used in the process of set occurrence selection to provide a range of values that will be used to locate the owner of a set occurrence.

2.2.5.3 Set Occurrence Selection – Each time DBCS references a particular set type implicitly during a STORE of an automatic member of that set type, DBCS must choose the occurrence of that set type into which it will insert the new record. You can specify one of two methods of set occurrence selection for a member record: CURRENT OF SET and LOCATION MODE OF OWNER.

If you specify CURRENT OF SET, DBCS chooses the current set occurrence (i.e., the one in which the CURRENT OF SET record belongs).

If you specify LOCATION MODE OF OWNER, DBCS selects the appropriate set occurrence by locating an owner record occurrence according to the location mode specified for the owner record. If the location mode of the owner is DIRECT, DBCS uses the DIRECT identifier 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. Note that the run-unit must initialize the DIRECT identifier or the CALC key in the UWA before referencing a set type. If the location mode of the owner is VIA set-name-2, DBCS must locate the owner of the set on the basis of its membership in set-name-2. 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, or an owner with a location mode of DIRECT or CALC, or the SYSTEM record. DBCS then calculates set occurrences back down the hierarchy until it reaches the owner originally specified. DBCS then stores the new record in relation to its owner record according to the SET ORDER. Note that in the case of LOCATION MODE OF OWNER, set order NEXT is equivalent to set order FIRST and that set order PRIOR is equivalent to set order LAST.

You should avoid set occurrence selection using LOCATION MODE OF OWNER unless you have a specific reason for using it. Such set occurrence selection can cause DBCS to unnecessarily search the data base for the correct owner. For instance, if the program were to store two member records one right after the other into the same set occurrence, DBCS would have to redundantly reselect the owner record during execution of the second STORE statement.

2.2.6 Describing Sub-Schemas

You use the Sub-schema DDL to define those areas, records, and sets from the schema that will be included in each sub-schema. Since an application program can only access a sub-schema, you are essentially limiting the program to those parts of the data base defined in the sub-schema. You can, however, copy all areas, records, and sets from the schema to a sub-schema, thus allowing a program to access the entire data base.

You can specify a privacy lock for a sub-schema. This lock is similar to the privacy locks for an area. It must be matched with a privacy key before a program can be compiled using that sub-schema.

Within the Sub-schema Record Entry, you can specify the data-items that are included in a data-aggregate. These can be COBOL data descriptions (level 03 and greater) or FORTRAN statements. Thus you can isolate the COBOL- and FORTRAN-specific data descriptions in each sub-schema.

Refer to Chapter 5 for a complete description of the syntax and rules for the sub-schema DDL.

2.2.7 Describing the Journal

A journal is a file in which a run-unit writes images of data base pages whenever it executes an updating DML command (STORE, DELETE, INSERT, REMOVE, MODIFY). There can be two kinds of images in a journal – BEFORE and AFTER. A BEFORE image is a copy of a data base page before it is changed. An AFTER image is a copy of a data base page after it is changed. At any time during a run-unit, the latest images in the journal match the pages in the data base. That is, DBCS always synchronizes writing to the journal with writing to the data base.

Although the images are the primary units of information in the journal, they can be grouped into larger units called commands or even larger units called transactions. A command contains all of the BEFORE/AFTER images that result from the execution of one updating DML command. Each command is delimited by a command header and trailer. A transaction is a user-specified group of commands. It is delimited by a transaction header and trailer. The size of the smallest transaction is one command, while the size of the largest possible transaction is the entire journal. The run-unit defines the bounds of a transaction by calling the JSTRAN and JETRAN subprograms to write the transaction headers and trailers. (See Section 2.3 in the *DBMS Programmer's Procedures Manual* for more information.)

You use the BEFORE images in the journal for backup/recovery of the data base either during a run-unit or during execution of the DBMEND program. (See Section 6.2 for information about DBMEND.)

You use AFTER images to bring an old copy of the data base up to date during the execution of the DBMEND program.

There is usually only one journal for each data base. You define its characteristics with DMCL statements. However, a run-unit can override the name you specified for the journal by means of a subprogram call. (See Section 2.3 in the *DBMS Programmer's Procedures Manual.*) The characteristics of the journal are its file specification, the kind of images in it, and its unit of backup/recovery (command or transaction).

2.2.7.1 Specifying Backup/Recovery – The CODASYL Data Base Task Group specification requires that DBMS provide automatic recovery of the data base if an error occurs during updating by a DML command. That is, if an error (exception) occurs while a DML command is updating the data base, DBMS must recover the data base by restoring it to the state it was in prior to the initiation of the command. DECsystem-20 DBMS uses the journal to provide the backup necessary for this recovery.

To perform backup, DBCS writes a command header and BEFORE images into the journal 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 a DML updating command, DBCS automatically restores the data base back to the last command header in the journal.

Support of this automatic recovery causes the run-unit to perform extra I/O. This is because it must write out all of its buffers into the journal as well as to the data base after each DML updating command is completed. This is to ensure that the journal and the data base are up to date and synchronized.

You can control the amount of backup/recovery and thus the amount of extra I/O by means of the DMCL IMAGES statement (see Chapter 3). With this statement you can specify that images in the journal are or are not ordered by command. If you specify ordering by command, DBCS performs the backup and automatic recovery described above. If you specify that images are not ordered by command, the run unit must perform recovery manually.

For a run-unit to perform manual recovery, it must create transactions by calling JSTRAN and JETRAN to write the transaction headers and trailers. If an exception occurs during updating, the run-unit must call the JBTRAN subprogram to restore the data base back to the last transaction header.

Backup using transactions can reduce the amount of extra I/O because DBCS will write out the journal and data base buffers only at the end of every transaction rather than at the end of every DML command. The most economical way, however, for a run-unit to perform I/O is to write the buffers only when necessary (i.e., only when a buffer must be overwritten to make room for a new data-base page). If you have a run-unit that cannot recover from an exception unless it is started again (from the beginning) you can specify that no recovery be performed during that run-unit and avoid the extra I/O. The run-unit should still write BEFORE images in the journal, however, so that you can use DBMEND to restore the data base.

2.2.8 Requesting Exception Interception

You can specify in the DMCL Environment Entry that you want DBCS to type a message and optionally exit when a particular kind of exception occurs. The note clause tells DBCS just to type a message. The INTERCEPT clause tells DBCS to type a message and exit to TOPS-10 command level; the run-unit can continue if the programmer types the TOPS-10 system CONTINUE command. The intercept facility gives you the ability to trace exceptions, debug new programs, or reduce the amount of procedural exception checking by the run-unit.

The kinds of exceptions that you can have noted and/or intercepted are BIND, CALL, UPDATE, SYSTEM, ALL, or UNANTICIPATED exceptions. BIND exceptions are those that can occur when DBCS binds the sub-schema to the run-unit. Binding occurs when the UWA is set up and the records from the data base are made accessible to the program. CALL exceptions are those that can occur when the run-unit calls a DBCS subprogram (e.g., SETDB, JSTRAN). UPDATE exceptions are those that can occur when a verb that updates the data base is executed (e.g., STORE, DELETE). SYSTEM exceptions are those that occur when there is a software error in DBCS or a data error in one of the files (data base, schema, or journal). ALL exceptions means all of the above plus exceptions that occur during execution of any other DML verb (e.g., IF, FIND). UNANTICIPATED means all of the above except 0307 and 0326.

You can specify both a NOTE and an INTERCEPT statement in the DMCL. Thus, you can note one class of exceptions (or all) and intercept another. Specifying NOTE ALL and INTERCEPT ALL, however, would be redundant, because INTERCEPT subsumes NOTE.

2.3 RUNNING THE SCHEMA PROGRAM

Once you have created the file containing the description of the schema, you must process it using the SCHEMA program. To do so, you must run SCHEMA and give it a command string as follows.

```
.R SCHEMA
*[output file-spec=]input file-spec [/switch]
```

The input file specification consists of the device, filename, and extension. If you omit the device, SCHEMA assumes DSK. You cannot omit the filename; but if you omit the extension, SCHEMA assumes .DDL. You can specify wildcarding in the input file specification.

The output file specification consists of the device, filename, and extension. If you omit the device, SCHEMA assumes that of the input file. If you omit the filename, SCHEMA assumes that of the input file. SCHEMA always uses the extension .SCH even if you specify a different one. If you omit the entire file specification. SCHEMA uses that of the input file except that it changes the extension to .SCH.

When you run SCHEMA, it creates not only the .SCH file, but also the .DBS files. As explained above, the .DBS files are the files containing the areas; thus, as a group, they are the data base. Two switches are provided that control whether or not the .DBS files are created. They are the /CREATE and /NOCREATE switches. If you do not include either switch, the .DBS files are created only if they do not already exist. If you include the /CREATE switch, SCHEMA always creates the .DBS files, even if they already exist. This will cause the previous version of the .DBS files to be superseded and to be lost to DBMS. If you include the /NOCREATE switch, the .DBS files are not created, even if they do not exist.

When SCHEMA first creates the .DBS files, they are zero-length. You must load them with data to have a functional data base. However, if you wish to rerun SCHEMA because you have made a change to the schema file, you will not wish to have the .DBS files created again unless you want to change and reload the data base. Thus, you would use the /NOCREATE switch (or no switch). When you do want to change the schema and the data base, you use the /CREATE switch to cause SCHEMA to write over each existing .DBS file with an empty file. To save the existing data, you should unload the data from the .DBS files to other files before you recreate them. Section 2.4 describes the changes to the schema that will and will not necessitate recreating the .DBS files.

The data definition process described in this Chapter is shown in Figure 2-7.

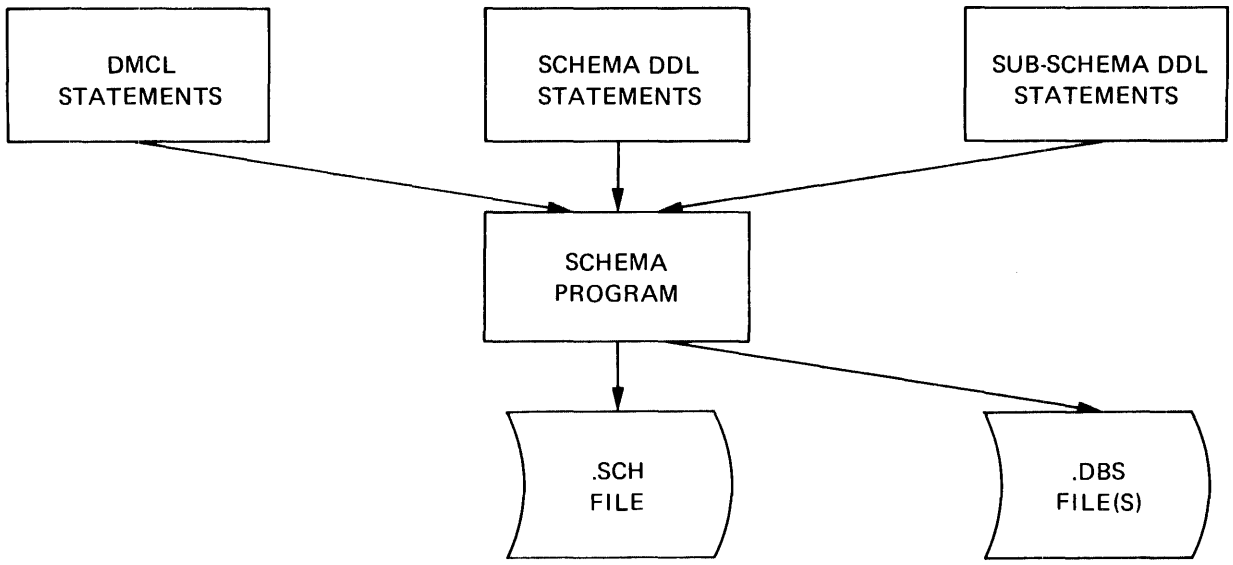


Figure 2-7 Data Definition Process

Every time you rerun SCHEMA and create a new .SCH file, SCHEMA changes the internal version number of the file. If an existing relocatable binary version of an application program accessing the schema is used, DBCS will issue an exception code of 1500 (see Appendix B of the *DBMS Programmer's Procedures Manual*) stating that a version discrepancy exists. This is because the REL file of the application program has the version of the schema that was available when the program was compiled. If the changes to the schema do not affect the portion of the data base that the program accesses, the run-unit need not be aborted. If this is not the case, the program should be recompiled.

2.4 CHANGING THE SCHEMA

There are two kinds of changes that you can make to the schema – those that require you just to rerun SCHEMA and those that require you to recreate the .DBS files and thus reload the data base as well. (Changes that require recompilation of the application programs are noted individually.)

The changes that do not require reloading of the data base are the following.

1. Any name currently in the schema. This may require changes in the application programs.
2. The buffer count.
3. An increased number of records per page.
4. The upper end of the range of an area; i.e., the number of the last page in the area. However, this change is meaningful only if you either have a no CALC records in the area or have a RANGE clause for each CALC record.
5. Any sub-schema. This will probably require changes in the application programs.
6. The kind of images in the journal.
7. Image ordering, i.e., whether or not images will be ordered by command.
8. Privacy locks. This may require changes to the application programs.
9. An area's temporary status.
10. Additional areas in a record's WITHIN clause. This may require changes in the application programs (i.e., they may have to add area-IDs).
11. New record types, but only after the last existing Record Entry. Otherwise, you would change the existing records' record type IDs.
12. Additional new set types as long as no existing records are used in them. This is because you would have to add new pointers to the existing records.

13. A record's set membership, i.e., from MANDATORY to OPTIONAL or vice versa and from AUTOMATIC to MANUAL or vice versa. This may affect the application program logic, so you should consider it carefully.
14. The form of set occurrence selection, i.e., from LOCATION MODE OF OWNER to CURRENT OF SET or vice versa.
15. The types of errors to be noted or intercepted.

Any change that requires a change in the record type ID, size, and division between pointers on one hand and data in a record on the other hand will require you to reload the data base. Other changes that would require reloading the data base are those that change existing set relationships, those that interfere with CALC-chains, or any changes to the description of the data-items in the records.

Although you can change neither the order of the record types as they appear in the schema nor their set relationships, you can add new record and set types. You must note the following restrictions, however, on adding new record and set types.

1. Add all new records to the schema at the end of the existing record types.
2. Add all new sets to the schema at the end of the existing set types.
3. Only new set types can contain new record types. Do not add existing record types to new set types.
4. Do not add new record types to existing set types.

2.5 USING SIMULTANEOUS UPDATE; DATA BASE DESIGN CONSIDERATIONS

As described in Section 2.2.1.2, DBMS allows a concurrent run-unit to update or retrieve data while another run-unit updates or retrieves in the same area. Concurrency is supported through use of the ENQUEUE/DEQUEUE facility of the TOPS-10 operating system. (Refer to Chapter 16 of the *DECsystem-10 Monitor Calls Manual* for a discussion of ENQUEUE/DEQUEUE.)

Note that this facility can be used to its best advantage for discrete programmer-initiated operations (for example, updates and interrogations). It is less efficient for generating reports that involve essentially sequential processing of sets.

A run-unit specifies how it intends to access data in the area with the syntax of the DML OPEN statement. (See Chapter 3 of the *DBMS Programmer's Procedures Manual* for a description of the Data Manipulation Language.) Depending on the concurrency it needs, a run-unit can choose one of three usage-modes allowing simultaneous update/retrieval:

UPDATE
PROTECTED UPDATE
RETRIEVAL

If a run-unit updates an area simultaneously with other run-units, it also updates the journal file. When you are using DBMEND, you must be able to isolate the changes each run-unit has made to the journal file. This is done through the mechanism of run-unit IDs.

Refer to Section 2.2.7, which describes the journal and to Chapter 6, which describes DBMEND.

Because sharing a journal involves DBCS in a certain amount of extra maintenance, it is important that each run-unit indicate to DBCS when it does not intend to share the journal. To do this, the programmer can specify the OPEN JOURNAL USAGE-MODE EXCLUSIVE UPDATE statement before opening any areas for update. (See Chapter 3 of the *DBMS Programmer's Procedures Manual* for a detailed description of the syntax.)

If the programmer has not specified an open-journal statement, the system simulates one depending on whether the area is being opened for update (shared) or for exclusive or protected update (exclusive).

The level of sharing (the resource shared) within the three simultaneous usage-modes (as maintained by ENQUEUE/DEQUEUE) is the data base. When a run-unit is executing an updating statement (and possibly sharing the journal file), other run-units are locked out of the data base. For this reason, use of the simultaneous-update usage-modes by run-units impacts the efficiency of DBMS. It's important, therefore, that run-units use the simultaneous-update modes in a discriminating way; in general, this means not unless such a mode is functionally necessary.

When a run-unit is within the framework of simultaneous update, then, its use of the data base as a resource is either exclusive or shared:

- Exclusive during execution of updating commands. This means that during the duration of an updating command no other run-unit can access the data base. (The updating commands are STORE, MODIFY, INSERT, REMOVE, and DELETE.)
- Exclusive during transactions if images are not ordered by command.
- Shared otherwise (FIND, GET, IF . . .).

This is true even when run-units are opening disjoint areas of the data base. If Run-unit A opens area 1 for update, for example, and Run-unit B opens area 2, Run-unit B will be locked out of the data base during the duration of execution of an updating command issued by Run-unit A.

You can influence the duration during which concurrent run-units are guaranteed exclusive control of the data base. You can do this with the IMAGES statement of the DMCL. (See Chapter 3.) If you specify that images are in order by command, this duration is the command. If you specify that images are not in order by command, this duration is the user-defined transaction (that is, a logical operation bounded by calls to JSTRAN and JETRAN). For efficient use of transactions, the application programmer should define transactions such that they are mainly calls to DBCS. A command executed outside of an entire transaction is, in effect, a default transaction. The duration of control over the data base – for a default transaction – is the command duration. Having default transactions facilitates certain types of application programs:

- those that only retrieve. They do not have to define transactions since they do not update the data base.
- those that want to define only a few transactions but otherwise want the minimum-duration control over the data base (and therefore minimum duration lockout of other run-units) otherwise provided by specifying IMAGES BY COMMAND.

If you omit the IMAGES statement, the system assumes that IMAGES are by command.

The following tables provide an overview of significant trade-offs you can make when deciding to use the simultaneous-update capability at your facility. The tables are intended to complement the text. Table 2-1 lists the six usages-modes and gives suggestions for using each.

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

RETRIEVAL[1]	You intend other run-units to open simultaneously with UPDATE or PROTECTED UPDATE.
UPDATE[1]	You intend other run-units to open simultaneously with UPDATE.
PROTECTED RETRIEVAL	You expect concurrent retrievers but no concurrent updaters.
PROTECTED UPDATE[1]	You intend other run-units to open with RETRIEVAL.
EXCLUSIVE RETRIEVAL	You really need this exclusiveness.
EXCLUSIVE UPDATE	You really need this exclusiveness.

[1] Simultaneous-update usage-mode.

Table 2-2 relates the level of simultaneity you can implement at your facility – and how you can achieve it – to overall design implications you must consider. The table is arranged in terms of levels of increasing duration of control of the data base resource by a run-unit when it opens an area in a simultaneous-update usage-mode.

Table 2-2
Levels of Simultaneity Related to Data Base Design Considerations

Level of Simultaneity	How to Achieve	Design Considerations
None	Use EXCLUSIVE RETRIEVAL/ EXCLUSIVE UPDATE usage-modes.	Inexpensive in terms of CPU time.
Simultaneous retrievers	Use PROTECTED RETRIEVAL usage-mode.	Inexpensive in terms of CPU time.
Pseudo-simultaneous updaters	Specify that run-units simultaneously open disjoint areas in EXCLUSIVE UPDATE usage-mode.	Inexpensive in terms of CPU time. It does, however, complicate the design/control problem. You must, for example, ensure that areas are truly disjoint and journals are properly managed at the applications level.
Begins Involvement of Simultaneous-Update Facility		
Duration of control of the data base resource is the command.	Specify IMAGES BY COMMAND. Optionally allow applications to define transactions.	Minimizes the duration during which a run-unit can have exclusive control of the data base (and thereby possibly lock out other run-units from the data base. (Those run-units outside the framework of simultaneous update must also consider efficiency implications related to the IMAGES statement.) See also Section 2.2.7.1.
Duration of control of the data base resource is the transaction plus default transactions.	Specify IMAGES NOT IN ORDER BY COMMAND. Ensure applications define transactions using JETRAN – JSTRAN only for those operations that must not have interference during execution. Other commands remain outside the scope of transactions.	Minimizes the duration during which a run-unit can have exclusive control of the data base while it allows definition of logical operations in terms of transactions. In effect, you have both command-duration and transaction-duration control over the data base.
Duration of control of the data base resource is the transaction.	Specify IMAGES NOT IN ORDER BY COMMAND. Ensure application programs divide all operations on the data base into logical units using JETRAN – JSTRAN. Ensure transactions are composed mainly of calls to DBCS.	Minimizes simultaneous-update CPU usage and is the most efficient for overall throughput. This is the most easily recoverable form of simultaneous-update usage; it is also the easiest to design because control issues are simple.

2.6 USING THE DBMS UTILITIES

DBMS provides three utility programs to help you utilize the data base. These utilities are DBINFO, DBMEND, and DAEMDB.

DBINFO is a utility program that gives you information about the contents of the data base. With DBINFO, you can get a dump of the data, a cross reference listing of the names in the data base, statistical information about the data base, and the amount of free space left in the data base. You can use DBINFO to provide documentation of the data base for yourself, your staff, and the programmers accessing the data base. DBINFO has the facility for limiting the data that it outputs to a specific sub-schema and a specific range of pages in an area. Thus, you need not give the information about the entire data base to those who only need to know about a portion of it. DBINFO is described in detail in Chapter 6.

DBMEND is the utility program that allows you to perform page recovery using the BEFORE images in the journal file. Although DBCS can use the journal file to automatically restore data base pages if an exception occurs during a run-unit (see Section 2.2.7.1), DBMEND is available if automatic restoration is not possible or too restricted. For example, DBCS cannot restore the data base from the journal when the run-unit terminates abnormally (e.g., crashes). In such cases you must use DBMEND to restore the data base. You can also use DBMEND to bring an old copy of the data base up-to-date by using a journal file with AFTER images. In addition to page recovery, DBMEND is useful for getting abstracts of the journal and for adjusting the Area Status Record of a .DBS file. For more information about DBMEND, see Chapter 6.

DAEMDB is a utility program that allows you to perform magnetic-tape journaling. The program copies data from the temporary journal file on disk (see Section 6.2.5.2) to magnetic tape. DAEMDB can be run under OPSER or as a normal time-sharing job from a terminal; it can control up to eight journals. Refer to Chapter 6 for a description of DAEMDB.

2.7 USING THE STATS SUBPROGRAM

STATS is a DBCS subprogram that a run-unit can call to obtain a printed summary of the status of the data base activity for each sub-schema.

The statistics produced are:

1. The cumulative run-time for each sub-schema.
2. The number of calls made to DBCS for each class of verb and the total of all calls to DBCS.
3. The number of data-base page accesses.
4. The number of times DBCS accessed a page twice or more in a row. This provides a measure of access locality.
5. The number of reads and writes performed on data base pages.
6. The number of times DBCS wrote a journal page.
7. The amount of run-time used for each call to DBCS (optional because it is costly to generate).

The amount of run-time used for each call to DBCS is shown as 0 on the printed summary unless you set the global symbol "STATS." to a nonzero value. One way you can accomplish this is by means of the /DEFINE switch to LINK. You can specify the "STATS." symbol as the argument to the switch. For example:

```
.R LINK
*prog-name, /DEFINE:STATS.:1, . . .
```

A COBOL program calls the STATS routine as follows:

```
ENTER MACRO STATS.
```

A FORTRAN program calls the STATS routine as follows:

```
CALL STATS
```

An example of the printed summary produced by STATS is as follows.

```
DBCS USAGE STATISTICS

STATISTICS FOR SUB-SCHEMA SUBS1
(CUMULATIVE RUNTIME    1817 MILLI-SECONDS)

CLASS                  CALLS      RUNTIME(MS)
DBCS-TOTAL               660                0
HOST                     100                0
CLOSE                      1                0
FIND                     197                0
GET                       99                0
INSERT                   100                0
OPEN                       1                0
STORE                    150                0
BIND                      11                0
CALL                       1                0
# DBPAGE ACCESSES                        1867
# OF SAME-DBPAGE ACCESSES                1592
# OF DBPAGE READS                          15
# OF DBPAGE WRITES                        165
# OF JOURNAL PAGE WRITES                  139
```

CHAPTER 3
THE DEVICE MEDIA CONTROL LANGUAGE (DMCL)

The Device Media Control Language (DMCL) enables you to select individual areas, assign them to files, and allocate storage for them. The DMCL is an extension to the Schema DDL, and thus, the DMCL statements must be executed by the DDL processor, SCHEMA. You include the DMCL entries in your schema file before the Schema DDL entries.

There are two types of DMCL entry that you can use to:

1. Specify parameters that apply to the schema as a whole, e.g., the number of records on a page and the name of the journal (Environment Entry).
2. Assign areas to files and specify the physical characteristics of these files (Area Entry).

For each area specified in the schema, you must have a DMCL Area Entry. On the other hand you can only specify one DMCL Environment Entry for each schema. The DMCL Environment Entry must precede the DMCL Area Entries.

The DMCL entries are described on the following pages. Each entry starts on a new page.

Certain conventions have been used in the descriptions of the DMCL and DDL statements in this chapter and Chapters 4 and 5. These conventions and their meanings are as follows.

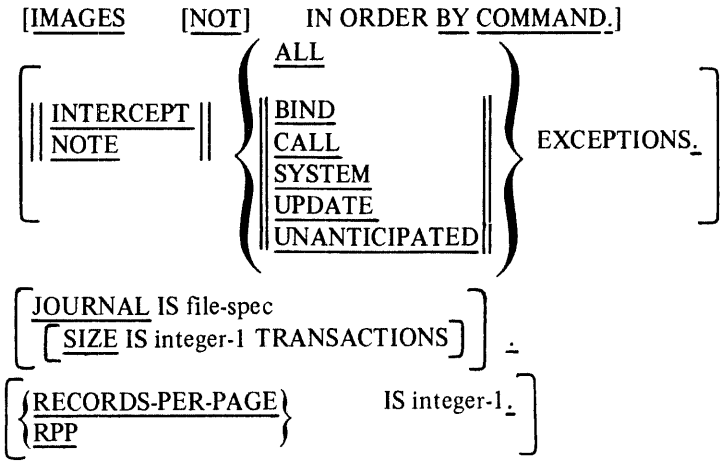
Lower-case characters	Information that you must supply (values, names, and other parameters).
Upper-case characters, underscored	Key words in the DMCL and DDL lexicons you must use when using the formats of which they are a part.
Upper-case characters, not underscored	Other words in the lexicons that serve only to make the 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 [ ]	A choice, optional. 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 DMCL and DDL 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.

DMCL ENVIRONMENT ENTRY

Function
Use the DMCL environment entry to specify those parameters that apply to the entire schema.

General Format



Technical Notes

1. The DMCL environment entry must precede all DMCL area entries.
2. The statements in the DMCL environment entry can appear in any order. They are shown here and described on succeeding pages in alphabetical order.
3. The entire DMCL environment entry can be omitted.

Example

```
IMAGES BY COMMAND.
NOTE ALL.
INTERCEPT BIND.
JOURNAL IS JRN:SCHAM
SIZE IS 1 TRANSACTIONS.
RPP 25.
```

IMAGES

Function

Use the IMAGES statement to specify whether DBCS or the run-unit defines the minimum unit of recovery of the data base.

General Format

[IMAGES [NOT] IN ORDER BY COMMAND.]

Technical Notes

1. IMAGES IN ORDER BY COMMAND means that DBCS will define commands as a unit of recovery during a run-unit. DBCS uses the journal's BEFORE images to perform the backup necessary for the recovery. A command in the journal is the group of BEFORE images associated with one DML updating command and delimited by a command header and trailer.
2. IMAGES IN ORDER BY COMMAND also means that if an exception occurs during execution of a DML updating command, DBCS will automatically restore the data base to its state prior to that command.
3. IMAGES IN ORDER BY COMMAND also defines the command as the duration of control of the data base within the framework of simultaneous update. See also Section 2.2.1.3.
4. IMAGES NOT IN ORDER BY COMMAND means that when the run-unit defines transactions they will be the unit of recovery during a run-unit. The run-unit uses the journal's BEFORE images to perform the backup necessary for the recovery. A transaction in the journal is the group of BEFORE images delimited by a transaction header and trailer. The run-unit defines the size of the transaction by means of calls to the JSTRAN and JETRAN subprograms, which write the transaction headers and trailers.
5. IMAGES NOT IN ORDER BY COMMAND also means that if an exception occurs during execution of a DML updating command, the run-unit must call the JBTRAN subprogram to restore the data base to its state prior to the transaction in which the exception occurred. In the framework of simultaneous-update, however, a command outside defined transactions does conform to the rules pertaining to IMAGES IN ORDER BY COMMAND.
6. IMAGES NOT IN ORDER BY COMMAND also defines the transaction as the duration of control of the data base within the framework of simultaneous update. See also Section 2.2.1.3.
7. If the run-unit does not define transactions in the journal and you have specified IMAGES NOT IN ORDER BY COMMAND, recovery cannot occur during the run-unit. Instead, you must run the DBMEND program to restore the data base using the BEFORE images in the journal. Refer to Section 6.2 for information about DBMEND.
8. Either you (in the DMCL BACKUP statement) or the run-unit (with a subprogram call) must specify BEFORE images in the journal if recovery is to be possible.
9. Refer to Section 2.2.7 for information about the journal and to Section 2.3 in the *DBMS Programmer's Procedures Manual* for information about the subprogram calls.
10. The IMAGES statement can appear anywhere in the DMCL environment entry.
11. If you omit the IMAGES statement, SCHEMA assumes IMAGES IN ORDER BY COMMAND.

Example

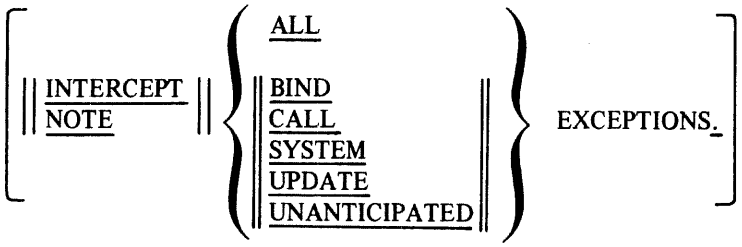
```
IMAGES BY COMMAND.
```

INTERCEPT/NOTE

Function

Use the INTERCEPT or NOTE statement to specify the exceptions that DBCS will intercept or note during a run-unit.

General Format



Technical Notes

1. INTERCEPT causes DBCS to type an error message and force the run-unit to exit to the monitor if an exception of the specified class occurs. The user of the run-unit can continue execution by typing the CONTINUE system command.
2. NOTE causes DBCS to type a message if an exception of the specified class occurs. DBCS does not stop the run-unit as it will when you specify INTERCEPT.
3. The classes of exceptions have the following meanings:
   a. BIND – exceptions that occur during binding of the sub-schema (i.e., when DBCS enters the SBIND, EBIND, BIND, and SETUSE routines).
   b. CALL – exceptions that occur during a call to one of the journalling, SETDB, or UNSET subprograms.
   c. SYSTEM – exceptions with a code number greater than 55. See Table B-2 of the *DBMS Programmer's Procedures Manual.*
   d. UPDATE – exceptions that occur during execution of the DML updating verbs (DELETE, INSERT, MODIFY, REMOVE, or STORE).
   e. UNANTICIPATED – all exceptions other than 0307 and 0326.
   f. ALL – all exceptions that can occur during the run-unit.
4. You can specify both the INTERCEPT and NOTE statements in the DMCL environment entry.
5. If you do not specify an INTERCEPT or NOTE statement DBCS will not intercept or note any exceptions that occur during the run-unit.
6. The INTERCEPT or NOTE statement can appear anywhere in the DMCL environment entry.

Example

```
NOTE ALL.
INTERCEPT BIND.
```

JOURNAL

Function
Use the JOURNAL statement to specify the file specification for the journal.

General Format

```
[ JOURNAL IS file-spec
  [SIZE IS integer-1 TRANSACTIONS] ] .
```

Technical Notes

1. The file specification is of the form:
   dev:filename[p,pn]
   where:
   dev: is the device. If you omit it, SCHEMA will use DSK:.
   filename is the filename of the journal. You cannot omit the filename for a disk file. At run-time, the device resolves either to disk or to magnetic tape.
   [p,pn] is the project programmer number. If you omit it, SCHEMA uses that of the logged-in user.

   SCHEMA appends a file extension of .JRN to the filename. Thus, you cannot specify a file extension.
2. The JOURNAL statement can appear anywhere in the DMCL environment entry.
3. If you omit the JOURNAL statement, SCHEMA creates a file specification in which the device is JRN, the filename is the schema-name, and the project programmer number is that of the logged-in user.
4. The SIZE clause allows you to control the size of a journal file.
5. The SIZE clause applies to any run-unit that has a disk journal which is not shared, and to any shared journal for which images have not been ordered by command (that is, they are ordered by transaction).
6. Integer-1 indicates the maximum number of transactions you want in the journal at any one time. After the number of transactions you specify, DBCS repositions the journal file pointer to the beginning of the journal file.

Example

```
JOURNAL IS JRN:SCHAM
SIZE IS 1 TRANSACTIONS.
```

RECORDS-PER-PAGE

Function

Use the RECORDS-PER-PAGE statement to specify the maximum number of records that can be stored on a page for all areas in the schema.

General Format

```
[ {RECORDS-PER-PAGE}      IS integer-1. ]
  {RPP               }
```

Technical Notes

1. Integer-1 is an unsigned positive decimal number in the range 2 through 511.
2. You can place this statement anywhere in the DMCL environment entry.
3. You can omit this statement as long as you include a RECORDS-PER-PAGE statement in each DMCL area entry.
4. If you specify the RECORDS-PER-PAGE statement in the DMCL environment entry and a RECORDS-PER-PAGE statement in a DMCL area entry, the statement in the DMCL area entry will override the statement in the DMCL environment entry.

Example

```
RPP 25.
```

DMCL AREA ENTRY

Function

Use the DMCL area entry to describe each area in the schema. You can specify the area-name, the number of records on a page, the number of buffers, the number of CALC-chains per page, the first and last pages, the page size, and the pages where a particular record type can be stored.

General Format

```
ASSIGN      [SYSTEM AREA]     area-name TO file-spec

[{RECORDS-PER-PAGE} IS integer-1]
 {RPP              }

 [BACKUP  ||BEFORE||     IMAGES]
          ||AFTER ||

[BUFFER COUNT IS integer-2]

[CALC AT MOST integer-3     {RECORDS-PER-PAGE}]
                            {RPP             }

FIRST PAGE IS integer-4

LAST PAGE IS integer-5

PAGE SIZE IS integer-6 WORDS

[RANGE OF record-name-1 IS PAGE integer-7 TO PAGE integer-8,
 [RANGE OF record-name-2 IS PAGE integer-9 TO PAGE integer-10] . . .].
```

Technical Notes

1. You must specify a DMCL area entry for each area in the schema.
2. The statements in the DMCL area entry are specified in the order shown, except that the RECORDS-PER-PAGE, BACKUP, BUFFER, and CALC statements can appear in any order between the ASSIGN and FIRST PAGE statements.
3. The individual clauses are described on the following pages in the order shown.

Example

```
ASSIGN PERSONNEL-AREA TO FILE1
RPP 50
BACKUP BEFORE IMAGES
BUFFER 4
CALC 10 RPP
FIRST PAGE 1
LAST PAGE 600
PAGE SIZE 512 WORDS
RANGE OF CUST IS 15 TO 30.
```

ASSIGN

Function

Use the ASSIGN statement to specify the file specification for an area. You can also specify that an area is the SYSTEM area.

General Format

ASSIGN [SYSTEM AREA] area-name TO file spec

Technical Notes

1. The ASSIGN statement must be the first entry in each DMCL area entry.
2. The area named in the ASSIGN statement must be an area defined in the schema.
3. The file specification is of the form:

   dev:filename[p,pn]

   where:

   dev: is the device. If you omit it, SCHEMA will use DSK:.

   filename is the filename for the area. You cannot omit the filename.

   [p,pn] is the project and programmer number. If you omit it, SCHEMA uses that of the logged-in user.

   SCHEMA appends a file extension of .DBS to the filename. The filename must be unique in the directory.
4. The SYSTEM AREA phrase causes the SYSTEM record to be stored in this area. If you do not specify the SYSTEM AREA phrase and you do specify OWNER IS SYSTEM at least once, SCHEMA uses the area specified in the area entry as the SYSTEM area. If you do not specify OWNER IS SYSTEM and do specify the SYSTEM AREA phrase, SCHEMA will ignore the SYSTEM AREA phrase.
5. Only one area entry can contain the SYSTEM AREA phrase.

Example

```
ASSIGN PERSONNEL-AREA TO FILE1
```

RECORDS-PER-PAGE

Function

Use the RECORDS-PER-PAGE statement to specify the maximum number of records that can be stored on a page for an individual area.

General Format

```
[ { RECORDS-PER-PAGE } IS integer-1 ]
  { RPP             }
```

Technical Notes

1. Integer-1 is a positive unsigned decimal number in the range 2 through 511.
2. The RECORDS-PER-PAGE statement can appear anywhere between the ASSIGN and FIRST PAGE statements in the DMCL area entry.
3. You can omit this statement from the DMCL area entry as long as you have specified a RECORDS-PER-PAGE statement in the DMCL environment entry.
4. If you specify a RECORDS-PER-PAGE statement in the DMCL environment entry and also in a DMCL area entry, the RECORDS-PER-PAGE statement in the DMCL area entry will override that in the DMCL environment entry.

Example

```
RPP 50
```

BACKUP

Function

Use the BACKUP statement to specify the kind of images that will be written into the journal during an updating run-unit.

General Format

Technical Notes

1. The BACKUP statement can appear anywhere in the DMCL area entry between the ASSIGN and FIRST PAGE statement.
2. The BACKUP statement causes DBCS to write BEFORE and/or AFTER images in the journal during a run-unit that updates the data base.
3. BEFORE images are copies of data base pages before updating is done. AFTER images are data base pages after updating is done.
4. If you want both BEFORE and AFTER images, you can specify both in a single BACKUP statement.
5. If you omit this statement or if a run-unit needs to change the kind of images, it can call subprograms to request the images it needs. Refer to Section 2.3 of the *DBMS Programmer's Procedures Manual.*

Example

```
BACKUP BEFORE IMAGES
```

BUFFER COUNT

Function
Use the BUFFER COUNT statement to specify the number of buffers for the area.

General Format

[BUFFER COUNT IS integer-2]

Technical Notes

1. Integer-2 is positive unsigned decimal number equal to or greater than 2.
2. The BUFFER COUNT statement can appear anywhere in the DMCL area entry between the ASSIGN and FIRST PAGE statements.
3. If you omit a BUFFER COUNT statement for an area, the default for that area is three buffers.

Example

```
BUFFER 4
```

CALC

Function
Use the CALC statement to specify the number of CALC-chain headers on each page of the area.

General Format

```
[CALC AT MOST integer-3   {RECORDS-PER-PAGE}]
                          {RPP              }
```

Technical Notes

1. Integer-3 is an unsigned positive decimal number equal to or greater than 0.
2. The CALC statement can appear anywhere in the DMCL area entry between the ASSIGN and FIRST PAGE statements.
3. By specifying the number of CALC-chain headers on each page, you can balance space overhead against access/store time for CALC records. That is, if you specify several CALC-chain headers, you will increase the size of the page header for the area but reduce the access/store time for CALC records. Conversely, if you specify few CALC-chain headers, you will reduce the size of the page header for the area, but may increase the access/store time for CALC records. If you do not have any CALC records that can be within an area, you should specify 0 as the number of CALC-chains per page. See Section 2.2.2 for more information.
4. If you omit the CALC statement, SCHEMA will use one CALC-chain for each page in the area.

Example

```
CALC 10 RPP
```

FIRST PAGE/LAST PAGE

Function
Use the FIRST PAGE and LAST PAGE statements to specify the number of the first and last pages of the area.

General Format

FIRST PAGE IS integer-4
LAST PAGE IS integer-5

Technical Notes

1. Integer-4 and integer-5 must be unsigned positive decimal numbers.
2. The FIRST PAGE and LAST PAGE statements must appear in the order shown. The FIRST PAGE statement must follow the ASSIGN statement and all of the RECORDS-PER-PAGE, BACKUP, BUFFER, and CALC statements.
3. The range of the page numbers specified for one area cannot overlap the range of page numbers of another area in the same data base. That is, if the range of one area is pages 1 through 120, no other area in the same data base can contain a page number less than 121. The numbering need not be continuous between areas. That is, if one area is numbered 1 through 120, another area's first page need not start at 121.
4. There is no default for the FIRST PAGE and LAST PAGE statements. You must specify them.

Example

```
FIRST PAGE 1
LAST PAGE 600
```

PAGE SIZE

Function

Use the PAGE SIZE statement to specify the amount of data that can go on a page. You will also be implicitly specifying the buffer size for the area, since the buffers for an area are set to the same number.

General Format

PAGE SIZE IS integer-6 WORDS

Technical Notes

1. Integer-6 is an unsigned positive decimal number.
2. The PAGE SIZE statement must appear after the LAST PAGE statement.
3. You can specify any number of words as the page size. However, because the TOPS-20 operating system works only with device pages, DBCS will derive a data base page size that is the smallest multiple of device page size if you do not specify such a value (i.e., if you specify a page size of 1000, DBCS will make the physical page size 1024).
4. The size you actually specify is called the logical page size. DBCS will not place more than this number of words of data on a data-base page. Having a logical page size that is less than the physical page has two benefits. One, it allows you to layer your data if you gradually increase the page size. Two, it allows you to (eventually) increase the record size without increasing the physical page size.
5. There is no intrinsic limitation on page size; it is suggested that page size not exceed 4096.

Example

```
PAGE SIZE 512 WORDS
```

RANGE

Function
Use the RANGE statement to specify the location in an area where a particular record can be stored.

General Format

```
[RANGE OF record-name-1 IS PAGE integer-7 TO PAGE integer-8,
 [RANGE OF record-name-2 IS PAGE integer-9 TO PAGE integer-10] . . .].
```

Technical Notes

1. Record-name-1, record-name-2 . . . are the names of record types that can be stored in this area.
2. Integer-7 through integer-10 . . . are unsigned positive decimal numbers. They specify valid page numbers in the particular area.
3. Integer-7 and integer-9 define the beginning of the range for a record type. Integer-8 and integer-10 define the end of the range for a record type.
4. The RANGE statement must appear after the PAGE SIZE statement.
5. You can use the RANGE statement to cluster record occurrences of the same type. If you specify it for CALC record types, you will also have the ability to increase the number of the last page of the area because the calc algorithm will use your specified range rather than the first/last page of the area. See Section 2.2.2 for more information.
6. You do not have to specify a RANGE statement for any record types. If you do not specify a RANGE statement for a record type, DBCS will treat the entire area as that record type's range.

Example

```
RANGE OF CUST IS 16 TO 30.
RANGE OF SALESMAN IS 26 TO 200.
```

CHAPTER 4

THE SCHEMA DATA DESCRIPTION LANGUAGE (DDL)

The Schema Data Description Language (DDL) enables you to describe the logical and physical mapping of a data base in terms of a schema. A schema written in the Schema DDL consists of four types of entry that serve to:

1. Identify the schema (Schema Entry);
2. Define areas (Area Entry);
3. Define records (Record Entry);
4. Define sets (Set Entry).

For each area, record type, and set type described in the schema, a separate entry is required. However, only one Schema Entry can appear in a schema. When you construct a schema, you must maintain the following ordering of the entries:

1. The Schema Entry must always be the first entry;
2. An Area Entry must precede the Record Entries for all record types within that area;
3. A Record Entry must precede the Set Entries for all record types that participate in those sets as either owners or members.

The Schema DDL entries are described on the following pages. Each entry begins on a new page.

SCHEMA ENTRY

Function
Use the schema entry to initiate the schema portion of a DDL program and to provide documentation.

General Format

SCHEMA NAME IS schema-name.

Technical Notes

1. The schema-name given here is for documentation only. The name of the schema file is taken from the command line you give to the SCHEMA program. All schema-names are limited to a maximum of six characters.
2. The schema identified by the specified schema-name consists of the DDL entries that appear after this entry and before the first sub-schema entry.

Example

SCHEMA NAME IS BARHEX.

SCHEMA AREA ENTRY

Function

Use the area entry to name an area, optionally specify privacy locks and usage-modes, and optionally specify that the area is temporary.

General Format

```
AREA NAME IS area-name-1
     [AREA IS TEMPORARY]
     [ PRIVACY LOCK [FOR [EXCLUSIVE  ] {UPDATE   } ] IS lock-1 ]...
                        [PROTECTED   ] {RETRIEVAL}
```

Technical Notes

1. All area names must be unique among names within the schema, and can be from 1 to 30 characters long.
2. You must specify at least one area-name in a schema. If you specify only one area-name, that area and the data base are, in effect, equivalent.
3. A temporary area is not shared among concurrent run-units. Any run-unit that opens an area defined as temporary is allocated a private unique occurrence of that area. This is true even when multiple run-units refer to the same area-name. When a temporary area is closed, record and set occurrences in the area are no longer accessible, and the space occupied by the temporary area is again available to the TOPS-10 file system.
4. PRIVACY LOCK specifies the privacy lock that applies to the use of an area. You can use a separate PRIVACY clause for each usage-mode. However, you should not specify the same usage-mode in more than one PRIVACY clause.
5. Lock-1 is a privacy lock that the run-unit must match with a privacy key, which is identical to the lock. Each lock must be a single alphanumeric name up to five characters long.
6. You can specify the same lock for one or more options included in the PRIVACY clause. If you omit the optional FOR clause, lock-1 applies to any use of the area.
7. If you do not specify a PRIVACY clause with one of the usage-modes, there is no restriction on who can open the described area with that usage-mode.
8. The privacy lock associated with a usage-mode must be satisfied by the run-unit so that it can open the area with that usage-mode.

Example

```
AREA NAME IS PERSONNEL-AREA
     PRIVACY LOCK EXCLUSIVE UPDATE IS PAXUP
     PRIVACY FOR RETRIEVAL IS PARER.
```

SCHEMA RECORD ENTRY

Function

Use the record entry to name a record type in the schema (and thus specify a generic name for all occurrences of the record type in the data base); and to give the characteristics of all record occurrences of that type within a data base.

General Format

```
RECORD NAME IS record-name-1

    LOCATION MODE IS { DIRECT identifier-1 [%pseudonym-1]                          }
                     { CALC USING data-name-1 [data-name-2] ...                    }
                     {     [DUPLICATES ARE [NOT] ALLOWED]                          }
                     { VIA set-name-1                                              }

    WITHIN area-name-1 [area-name-2 ... AREA-ID IS identifier-2 [%pseudonym-2] ].

02 data-name-3 [%pseudonym-3] { PICTURE ... } [OCCURS ...].
                              { SIZE ...    }
                              { TYPE ...    }
```

Technical Notes

1. You must use a separate record entry to describe each record type in the schema.
2. A record entry must precede the set entries for all record types that participate in those sets as either owners or members.
3. For each record declared in a record entry, SCHEMA creates a record type ID. This record type ID is a number that is assigned in sequence, i.e., 33, 34, 35 . . . (DBCS uses the first 32 record type IDs, see Appendix C), according to the order in which you declare each record. Thus, the first record declared has a record type ID of 33 and the fifth record declared has a record type ID of 37. SCHEMA uses the record type IDs as major sort keys when you declare that ORDER IS SORTED. Refer to the ORDER clause for information about ORDER IS SORTED.
4. Each of the clauses in the record entry is described separately on one of the following pages.

Example

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

    02 ACCOUNT PIC X(6).
```

RECORD NAME

Function
Use the RECORD NAME clause to specify the name of the record type.

General Format

RECORD NAME IS record-name-1

Technical Notes

1. Each record name must be unique among the names in a schema, and can be 1 to 30 characters long.

Example

```
RECORD NAME IS CUSTOMER-RECORD
```

LOCATION MODE

Function

Use the LOCATION MODE clause to define how a record occurrence is physically stored in the data base, and to define the set occurrences into which a record occurrence is stored when the set occurrence selection is LOCATION MODE OF OWNER.

General Format

```
LOCATION MODE IS { DIRECT identifier-1 [%pseudonym-1]                    }
                 { CALC USING data-name-1 [data-name-2] ...               }
                 {     [DUPLICATES ARE [NOT] ALLOWED]                     }
                 { VIA set-name-1                                         }
```

Technical Notes

1. Identifier-1 is a variable used to hold a database key. It is not part of a record, but will appear in the UWA. Pseudonym-1, if specified, must be preceded by a percent sign. It can contain up to six characters and cannot contain hyphens. You should provide a pseudonym for FORTRAN use when identifier-1 contains more than six characters and/or hyphens.
2. Data-name-1 and data-name-2 must refer to data-items included in the record being described.
3. Set-name-1 must be a set in which the record is defined as a member.
4. You use the LOCATION MODE clause to control placement of records according to the options selected. DIRECT or CALC causes placement to be controlled by identifier-1 or by data-name-1, 2 . . . respectively. VIA set-name-1 causes placement to be as close as possible to the logical insert point of the record in set-name-1 thus permitting clustering of data. (See also Section 2.2.3.2.)
   a. The DIRECT option directs DBCS to physically store the record on the same page as the record that is current of area if identifier-1 is initialized to 0. If identifier-1 is not 0, DBCS attempts to store the record according to the page number of the database key in identifier-1. In other words, by placing a database key in identifier-1, the run-unit directs DBCS to place the record on the first empty line on the page specified.
   b. The CALC option directs DBCS to physically store the record according to the CALC keys (i.e., data-name-1, data-name-2 . . .). Using these keys, DBCS calculates a page and stores the record in the first empty line on the page. The run-unit can retrieve the record by specifying the correct CALC keys in data-name-1, data-name-2 . . . DBCS calculates the page and then searches the CALC-chain until it finds the record.
   c. The VIA set-name-1 option directs DBCS to physically store the record as close as possible to the record it will (probably) be NEXT OF after DBCS applies the set type's set order.
5. DBCS uses the LOCATION MODE clause logically when the set occurrence selection specified in a MEMBER statement is LOCATION MODE OF OWNER. Refer to the SET OCCURRENCE SELECTION clause for further information.
6. The DUPLICATE clause refers to the CALC keys. If you specify that DUPLICATES ARE NOT ALLOWED, DBCS returns exception 1205 if an attempt is made to store a record whose CALC key values are identical to those of an occurrence already in the data base. When you omit this clause, the default is DUPLICATES ARE ALLOWED. However, if record-name-1 owns any set in which SET OCCURRENCE SELECTION is LOCATION MODE OF OWNER, you must specify DUPLICATES NOT ALLOWED for its CALC keys.

Example

```
LOCATION MODE IS CALC USING ACCOUNT
```

WITHIN

Function
Use the WITHIN clause to tell DBCS the area where occurrences of the record will be allowed.

General Format

WITHIN area-1 [area-name-2 . . . AREA-ID IS identifier-2 [%pseudonym-2]].

Technical Notes

1. Area-name-1, area-name-2 must be the names of areas that you have already defined in a schema DDL area entry. If present, identifier-2 will cause the INVOKE processor to allocate a variable of that name in the UWA. This variable (or its pseudonym) will be PIC X(30) USAGE DISPLAY-7 in COBOL and INTEGER (5) in FORTRAN. See also Table 4-1 for the usage-modes in the schema declarations and in FORTRAN and COBOL.
2. When you specify more than one area-name, the contents of identifier-2 at the time of a DML STORE command determine the area into which a record occurrence is placed. Also, when the run-unit uses FIND rse 5 (FIND for CALC records), the area-ID tells DBCS the area in which to apply the CALC algorithm.
3. The pseudonym must be preceded by a percent sign. It can contain up to six characters and cannot contain hyphens. You should provide a pseudonym for FORTRAN use when the AREA-ID identifier contains more than six characters and/or hyphens.
4. The run-unit must initialize identifier-2 with an appropriate area-name prior to the execution of a STORE command. Identifier-2 can be initialized to 0 for records with location mode VIA. If identifier-2 is initialized with 0, DBCS stores that record occurrence in the area of the record occurrence that the new record will be NEXT OF, provided that record can be stored in that area.

Example

```
WITHIN MARKETING-AREA
```

SCHEMA DATA ENTRY

Function
Use the data entry to name a data-item or data-aggregate.

General Format

Format 1

```
02 data-name-3 [%pseudonym-3] { PICTURE } IS picture-string
                              { PIC     }
   [ USAGE IS  DISPLAY   ]
               DISPLAY-6
               DISPLAY-7   [OCCURS integer-1 TIMES].
               DISPLAY-9   ]
```

Format 2

```
02 data-name-3 [%pseudonym-3] SIZE IS integer-2 { WORDS                          }
                                                { USAGE { DISPLAY   }            }
                                                        { DISPLAY-6 }            }
                                                        { DISPLAY-7 }            }
                                                        { DISPLAY-9 }            }

   [OCCURS integer-1 TIMES].
```

Format 3

```
02 data-name-3 [%pseudonym-3]  TYPE IS { [ FLOAT ] [ { DECIMAL } ] [ REAL    ] }
                                       { [ FIXED ] [ { DEC     } ] [ COMPLEX ] }
                                       { [       ] [ { BINARY  } ] [         ] }
                                       { [       ] [ { BIN     } ] [         ] }
                                       { DBKEY                                  }
   [integer-3] [, integer-4] }  [OCCURS integer-1 TIMES].
```

Technical Notes

1. A data-name is a name that is not identical to any reserved word (see Appendix A), and may be up to 30 characters long. A data-name must be unique within a schema and cannot be the same as any data-name in a program that accesses the data base.
2. A pseudonym, if present, must be preceded by a percent sign (%). It can be up to six characters long and cannot contain hyphens. You should provide the pseudonym for FORTRAN use when the data-name contains more than six characters and/or hyphens.
3. A data entry names and describes a numeric or alphanumeric data-item, or allocates space for a data-aggregate.
4. You can use Format 1 to describe elementary alphanumeric data-items that are not part of a data-aggregate. The 02 level number is required; the picture string must conform to the rules of ANSI COBOL picture-strings, but it can only contain the symbols S9AXPV and ( ).

By specifying the USAGE phrase, you can describe the usage-mode of the alphanumeric data – SIXBIT (DISPLAY or DISPLAY-6), ASCII (DISPLAY-7), or EBCDIC (DISPLAY-9). Table 4-1 shows the allocation mechanism for FORTRAN and COBOL for each usage-mode described in the schema.

Table 4-1
Usage-Modes for FORTRAN and COBOL

Schema Declaration	No. of Characters in Field	FORTRAN Declaration	COBOL Declaration
DISPLAY	N	INTEGER (N/5)	DISPLAY-6 PIC X(N)
DISPLAY-6	N	INTEGER (N/5)	Same as schema
DISPLAY-7	N	INTEGER (N/5)	Same as schema
DISPLAY-9	N	INTEGER (N/4)	Same as schema
Note: If the value of the expression in the FORTRAN usage-mode is a fraction, it is always rounded up to the next whole number (e.g., 7/5 = 2).			

5. You can use Format 2 to describe a data-aggregate. When you specify the WORDS phrase, you declare the number of 36-bit computer words that will hold the data-aggregate. When you specify the USAGE phrase, you declare the representation of the data-aggregate. The size of each character is related to the usage-mode selected. DISPLAY or DISPLAY-6 (SIXBIT) means 6-bit characters, six to a 36-bit word; DISPLAY-7 (ASCII) means 7-bit characters, five to a 36-bit word; and DISPLAY-9 (EBCDIC) means 9-bit characters, four to a 36-bit word.
   To further describe a data-aggregate, you can include text with it in its sub-schema declarations. This text will be supplied to the run-unit when it invokes that sub-schema.
6. You can use Format 3 to describe elementary numeric data or database keys. Only certain combinations of numeric keywords are supported; they are shown in Table 4-2. If you specify an unsupported combination (e.g., FLOAT DECIMAL COMPLEX), SCHEMA will give you an error message. If you do not specify one of each of the numeric keyword pairs, SCHEMA uses FIXED, BINARY, and REAL as the defaults.
   You can specify integer-3 as the precision of the data-item. The precision is treated as decimal or binary digits depending on whether you have specified BINARY or DECIMAL. Table 4-2 gives the default precisions for the supported combinations of numeric types. Not every supported combination of numeric keywords is intrinsically supported in both host languages. Table 4-2 shows the numeric types as they are declared for the host languages. Note the COBOL precisions for BINARY numeric types are derived from Table 4-3.
   When you specify FIXED BINARY REAL for COBOL programs, you should use either 35 (single) or 70 (double) as the precision. If you use a different precision, the COBOL object-time system will have to perform extra work to keep your specified precision, and left-most truncation may occur if the data-item is carelessly moved or used in computations.
   To specify a scale factor for a fixed-point number, you can include integer-4. A scale factor indicates that the internal form of the number is different from the external form by some number of powers-of-10. That is, the decimal point is moved to the right or left of the internal number depending on whether the scale factor is positive or negative. For example +6 means that the decimal point is moved six places to the right on the internal number and -6 means that the decimal point is moved six places to the left on the internal number. If you do not specify a scale factor, SCHEMA assumes that there is no scale factor.
   If you specify TYPE DBKEY, SCHEMA generates a data-item that is INTEGER in FORTRAN and USAGE DATABASE-KEY in COBOL. The data-item declared as TYPE DBKEY can be used to store a database key.

Table 4-2
Numeric Types for FORTRAN and COBOL

Schema Declaration	Precision Range (Bits)	Default Precision (Bits)	FORTRAN type	COBOL 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)

Table 4-3
Relation Between Binary and Decimal Precision

Binary Precision declared in Schema	Decimal Precision in COBOL
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)
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)

7. You can use the OCCURS clause to declare that the data item/aggregate is an array (or table).

Example

```
02 ACCOUNT PIC X (12) USAGE DISPLAY-7.
```

SCHEMA SET ENTRY

Function

Use the set entry to describe a set type in the schema. This description will be used by DBCS to store and find record occurrences in the set occurrence of this type.

General Format

```
SET NAME IS set-name-1
    MODE IS CHAIN [LINKED TO PRIOR]
    ORDER IS { ALWAYS { FIRST | LAST | NEXT | PRIOR }
             { SORTED [ WITHIN RECORD-NAME
                      | BY DATABASE-KEY
                      | DUPLICATES ARE [ FIRST | LAST | NOT ] ALLOWED ] }
    OWNER IS { record-name-1 | SYSTEM } [.]
```

Technical Notes

1. You must include a set entry for each set in the schema.
2. You must define the records described in the set entry before you define the set entry.
3. You can specify the clauses in the set entry in any order.
4. You can optionally end the set entry with a period.
5. Each of the clauses in the set entry is described on a separate page below.

Example

```
SET NAME IS ORDER-SET
    MODE IS CHAIN LINKED TO PRIOR
    ORDER IS ALWAYS LAST
    OWNER IS CUSTOMER-RECORD.
```

SET NAME

Function
Use the SET NAME clause to specify the generic name for all occurrences of the set type in the data base.

General Format

SET NAME IS set-name-1

Technical Notes

1. Each set-name must be unique among the names of the schema, and can contain up to 30 characters.

Example

```
SET NAME IS ORDER-SET
```

MODE

Function
Use the MODE clause to specify the mechanism for the manipulation of a set.

General Format

MODE IS CHAIN [LINKED TO PRIOR]

Technical Notes

1. Each set within a schema can have only one MODE clause. All participating records of a set are linked to the next record.
2. The optional LINKED TO PRIOR clause causes DBCS to generate an additional pointer in the prior direction for the owner record and for each member record of each occurrence of the set.
3. DBCS automatically adds PRIOR pointers to members of sorted sets. See the ORDER clause for more information about sorted sets.
4. If the set order is ORDER IS LAST, the OWNER record always contains a PRIOR pointer.

Example

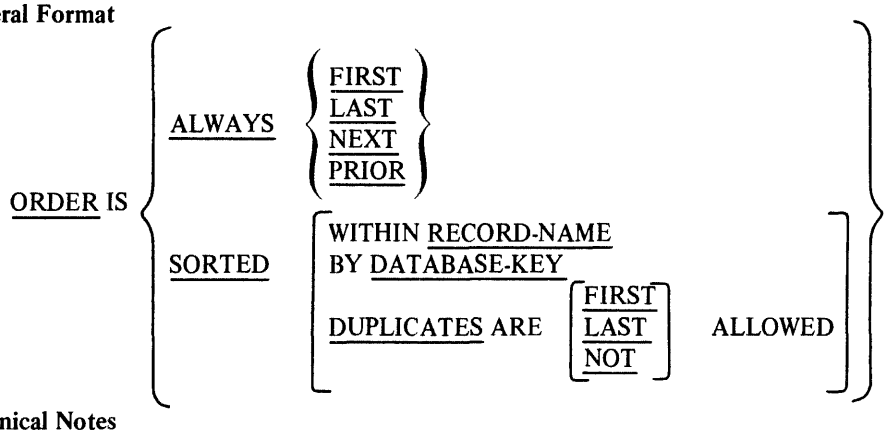
```
MODE IS CHAIN LINKED TO PRIOR
```

ORDER

Function

Use the ORDER clause to specify the insertion point of a member record occurrence within a set occurrence and thereby define the order of logical linkage.

General Format



Technical Notes

1. ORDER FIRST refers to the position within the set occurrence that immediately follows the owner record occurrence. Thus, the newest member record inserted into the set occurrence becomes the first member of the set occurrence.
2. ORDER LAST refers to the position within the set occurrence that immediately precedes the owner record occurrence. Thus, the newest member record occurrence becomes the last member in the set occurrence.
3. When you specify ORDER IS LAST and do not specify LINKED TO PRIOR, the owner record of each set occurrence will have a pointer to the last member record of that set occurrence.
4. ORDER PRIOR/NEXT refer to insertion points relative to the current record of the set. If the SET OCCURRENCE SELECTION is LOCATION MODE OF OWNER, ORDER IS PRIOR/NEXT is equivalent to ORDER IS FIRST/LAST.
5. The ORDER IS SORTED clause allows you to specify a data-dependent set order. However, this clause closely interacts with the ASCENDING/DESCENDING phrase of the MEMBER clause, which is described later in this chapter. Table 4-4 shows the relationship between the ORDER and MEMBER clauses depending on the syntax chosen for each.
   a. If you specify the ORDER IS SORTED clause, but do not use the WITHIN RECORD-NAME, BY DATABASE-KEY, or DUPLICATES phrase, the record type ID of each member record type is used as its major sort key. (See the Record Entry for a description of record type IDs.) Since the record type IDs are assigned sequentially on a first-come basis, the record types might not be sorted alphabetically. You can force an alphabetical sort by declaring the record types in alphabetical order by their names. That is, the first record-name in alphabetical order should be in the first Record Entry and so forth. You specify minor sort keys by the ASCENDING/DESCENDING phrase with the DUPLICATES option required for each member record type. But, if you do not use the ASCENDING/DESCENDING phrase for a member record type, the database key of each occurrence of that record type is used as its minor sort key. (This is not recommended.)

b. The optional WITHIN RECORD-NAME phrase causes each member record's occurrences to be sorted without regard to the record occurrences of other record types in the set occurrence. This does not mean that there is an implied major sort by record type. It means that when a given type of record is considered independently of any other member record-type, it is in sequence by its own sort key(s). If you do not use the ASCENDING/DESCENDING phrase with the DUPLICATES option for a member record type, the database keys of the occurrences of that record type are used as its ascending sort keys. (This is not recommended.)

c. The optional BY DATABASE-KEY phrase specifies that the member records of a set occurrence are kept in ascending (record-name independent) sequence by their database keys. This and the other cases of database keys used as sort keys should be avoided unless you expect single set occurrences to densely (i.e., several record occurrences per page) cover several data base pages. In such a case, FIND NEXT will operate more efficiently.

d. Use of the optional DUPLICATES phrase means that the member records in a set occurrence are to be maintained in a single sequence regardless of the number of different member record types specified in the set entry. You specify the common sort key(s) in the ASCENDING/DESCENDING phrases for each record type; they should agree in size and mode (however, SCHEMA does not enforce this). The DUPLICATES phrase specifies the action to be taken when a new record occurrence is to be added to the set and the values of its sort control data-items are duplicates of sort control data-items of a record occurrence that currently participates as a member of the set occurrence. For sets with only one member, this is the type ORDER SORTED to specify.

Example

```
ORDER IS ALWAYS LAST
```

Table 4-4 Relationships Between ORDER and MEMBER Clauses

SCHEMA Syntax Chosen				Effect of Syntax Chosen		
ORDER Clause Specified in SCHEMA	**ASCENDING/DESCENDING on MEMBER Clause**		**DUPLICATES on MEMBER Clause**	**Major Sorting/Ordering Sequence**	**Minor Sorting/Ordering Sequence**	**Overview of Ordering in Set Occurrence**
ORDER IS ALWAYS [FIRST, LAST, NEXT, PRIOR]	Not allowed		N/A	None	None	Not sorted
ORDER IS SORTED	Allowed for each member record type	If ASCENDING/ DESCENDING present on MEMBER Clause, then→	Required	Major sort key is member record type as ordered previously in user's schema	Minor sort keys as given by user in MEMBER clause	Hierarchically keyed lists, i.e., a list of lists with minor keys as specified by user.
		If ASCENDING/ DESCENDING not present on MEMBER Clause, then→	N/A		Database key for each member record occurrence	Hierarchically ordered lists in sequence within member-record type. (Minor lists are in ascending sequence by database key.)
ORDER IS SORTED WITHIN RECORD-NAME	Allowed for each member record type	If ASCENDING/ DESCENDING present on MEMBER Clause, then→	Required	None. Note that member-record occurrences ordered within record type are in effect ordered *by* record type.	Within each record type, sort keys as specified by user in member clause	Several independent keyed lists within the set occurrence.
		If ASCENDING/ DESCENDING not present on MEMBER Clause, then→	N/A		Database key for each member record occurrence within its own record type within the set occurrence.	Several independent ordered lists within the set occurrence (in ascending sequence by database key).
ORDER IS SORTED BY DATABASE-KEY	Not allowed		N/A	Database key for all member occurrences independent of record type.	None	A unified ordered list for the entire set occurrence (in ascending sequence by database-key).
ORDER IS SORTED DUPLICATES ARE [FIRST / LAST / NOT] ALLOWED	Required on each member clause for each member record type. These are the common sort keys that must agree among all record types on size and mode.		Not allowed	Only one sort sequence for entire set occurrence is maintained. Major sort keys are as specified by user in ASCENDING/DESCENDING clauses for members.	None	A unified list keyed by a common member-record key. This is suggested for single member sets.

OWNER

Function

Use the OWNER clause to specify the name of the owner record or to specify that the owner record of the set is the system.

General Format

```
OWNER IS    { record-name-1 }
            { SYSTEM        }
```

Technical Notes

1. Record-name-1 must be previously defined in a record entry.
2. When you name a record as the owner, you are specifying that each occurrence of that record will establish an occurrence of the set to which it belongs.
3. When you specify OWNER IS SYSTEM, you are specifying an inherently singular set; i.e., one that has a single occurrence. Because of this, certain efficiencies result (e.g., the owner record can always be located quickly). See the DMCL Area entry for information on specifying the SYSTEM area.
4. If the LOCATION MODE of record-name-1 is VIA, you must define the set referenced in the VIA phrase before you define any sets in which record-name-1 is OWNER.

SCHEMA MEMBER ENTRY

Function

Use the MEMBER entry to name each record type that can be a member of the set named and to specify the type of membership in that set.

General Format

```
MEMBER IS record-name-1   {MANDATORY}  {AUTOMATIC}
                          {OPTIONAL }  {MANUAL   }
    [LINKED TO OWNER]
[{ASCENDING                }     KEY IS data-name-1 [data-name-2] . . .
 {DESCENDING               }
 {ASCENDING  RANGE         }
 {DESCENDING RANGE         }]
    [DUPLICATES ARE [FIRST]  ALLOWED]
                    [LAST ]
                    [NOT  ]
  [SET OCCURRENCE SELECTION IS THRU  {CURRENT OF SET          }]
                                     {LOCATION MODE OF OWNER  }
[{USING data-name-2 [data-name-3]. . .                                  }]  [.]
 {ALIAS FOR data-name-4 IS identifier-1 [%pseudonym-1]. . .             }
```

Technical Notes

1. Record-name-1 must be previously defined in a record entry.
2. Record-name-1 cannot be the name of the record specified in the OWNER clause of the set entry to which this member entry belongs.
3. You must specify a MEMBER clause for each record type that can participate as a member in the set being described. You can declare more than one record type as a member of any given set.
4. You can define a record as a member in more than one set.
5. Each occurrence of a member record can participate in only one occurrence of a particular set type. That is, within a set, each member record occurrence may not have more than one owner.
6. You cannot define a record as both an owner and a member of sets such that a cycle is formed in which all records participate as automatic members in the sets included in the cycle.
7. Each of the phrases used to describe the member record types is described separately on the following pages.

Example

```
MEMBER IS ORDER-RECORD OPTIONAL MANUAL
```

MEMBERSHIP

Function

Use the MEMBERSHIP phrases to specify the ways in which occurrences of the record type can be added and removed from occurrences of the set type.

General Format

MEMBER IS record-name-1 { MANDATORY / OPTIONAL } { AUTOMATIC / MANUAL }

Technical Notes

1. AUTOMATIC/MANUAL refers to the way in which an occurrence of the record type is added to an occurrence of the set type. MANDATORY/OPTIONAL refers to whether an occurrence of the record type can be removed from an occurrence of the set type by means of the DML REMOVE command.
2. If you use AUTOMATIC, an occurrence of record-name-1 is inserted into the selected occurrence of the set at the time the record occurrence is stored.
3. If you use MANUAL, an occurrence of record-name-1 can only be inserted into an occurrence of the set by means of the DML INSERT command.
4. If you use MANDATORY, the run-unit cannot use the DML REMOVE command to remove an occurrence of record-name-1 from an occurrence of the set. The record occurrence can be deleted from the data base by means of the DELETE command. It will be deleted from the data base if its owner is deleted.
5. If you use OPTIONAL, an occurrence of record-name-1 can be removed from an occurrence of the set by means of the DML REMOVE command.

Example

```
MEMBER IS ORDER-RECORD AUTOMATIC OPTIONAL
```

LINKED TO OWNER

Function

The LINKED TO OWNER phrase causes each member record occurrence of this record type to contain a pointer to its owner.

General Format

[LINKED TO OWNER]

Technical Notes

1. This phrase is optional. If you do not specify it, the member record occurrences of this record type will not be directly linked to the owner record occurrence of this set type. However, the owner record will always be indirectly accessible through the forward linkages in the set. Note that you need not specify LINKED TO OWNER if the owner is SYSTEM because DBCS can always locate the SYSTEM record quickly.

Example

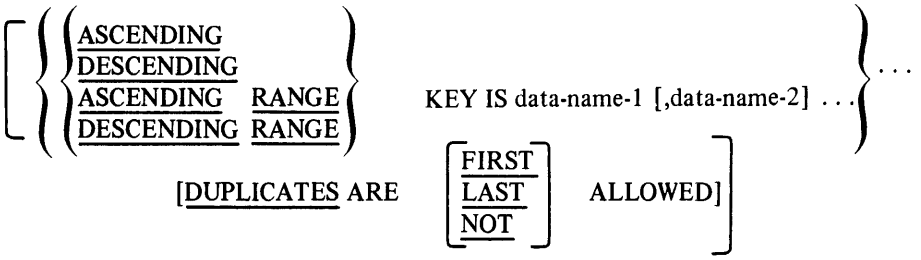
LINKED TO OWNER

ASCENDING/DESCENDING

Function

Use the ASCENDING/DESCENDING phrase to specify the sort control keys for the member records of a sorted set. You can also specify the action to be taken when a new record occurrence is to be added to the set whose sort key values are duplicates of sort key values of a record occurrence that currently participates as a member of the set occurrence.

General Format



Technical Notes

1. Data-name-1, data-name-2. . . must refer to data-items specified in the record entry for this member record. The data-items as a group are the sort key; each individual data-item is a sub-key. The order in which you specify the keys will define the major to minor sequence for sorting.
2. Within a member record, you can subdivide a sort key into subkeys. You can define some of the sub-keys as ascending keys and others as descending keys.
3. If you have defined multiple member record types for a set, the keys of different record types are completely independent unless you have specified ORDER SORTED DUPLICATES ALLOWED. Also, you can intermix the ASCENDING/DESCENDING KEY phrases among record types. That is, you can define one record type with an ASCENDING KEY phrase and another with a DESCENDING KEY phrase.
4. If you use the DUPLICATES ARE NOT ALLOWED phrase, DBCS will reject the insertion into any given set occurrence of member record occurrences with duplicate values for the specified ascending/ descending keys. This may occur when DBCS attempts to store a new record occurrence in the data base, insert an existing record occurrence into a set, or modify the value of a data item specified in an ASCENDING or DESCENDING KEY phrase.
5. If you use the DUPLICATES ARE FIRST or the DUPLICATES ARE LAST phrase, member record occurrences with duplicate values for the specified ascending/descending keys will be inserted by DBCS respectively before or after any existing member occurrence with such duplicate values.
6. If you do not include any of the optional words FIRST, LAST or NOT in the DUPLICATES phrase, the insertion point of duplicate member record occurrences relative to existing duplicates is unpredictable.
7. If you specify the keyword RANGE for an ascending or descending key, all following keys will also be treated as range keys.
8. Specifying the optional keyword RANGE does not affect the way in which key(s) are used in the process of maintaining set order. The effect of range keys occurs during set occurrence selection and is discussed in the description of that phrase.

Example

```
ASCENDING KEY IS RATE DUPLICATES ARE FIRST
```

SET OCCURRENCE SELECTION

Function

Use the SET OCCURRENCE SELECTION phrase to define the rules governing the selection of the appropriate occurrence of a set for the purpose of inserting an automatic member record during the execution of a STORE statement.

General Format

```
[SET OCCURRENCE SELECTION IS THRU  {CURRENT OF SET
                                   {LOCATION MODE OF OWNER

    [{USING data-name-2 [data-name-3]. . .                          }]]
    [{ALIAS FOR data-name-4 IS identifier-1 [%pseudonym-1]. . .     }]]  .
```

Technical Notes

1. The SET OCCURRENCE SELECTION phrase for the appropriate member record will govern the selection of the specific set occurrences when a STORE command is executed and the object record is an AUTOMATIC member of the sets.
2. If you do not explicitly specify a SET OCCURRENCE SELECTION phrase for a record, SCHEMA will assume SET OCCURRENCE SELECTION CURRENT OF SET. If the owner of the set is SYSTEM, you cannot specify a SET OCCURRENCE SELECTION phrase.
3. All data-names (except for the DIRECT key) must refer to declared data-items of the owner record of the set the clause is imbedded within.
4. The CURRENT OF SET option causes DBCS to select the current occurrence of the set named in the set entry (set-name-1) as the set occurrence into which to insert the (AUTOMATIC) member occurrence that is being stored. In other words, DBCS assumes that the set occurrence to be used has already been procedurally pre-selected by the run-unit.
5. When you specify the LOCATION MODE OF OWNER option, DBCS will choose the appropriate set and record occurrences by using the values of various UWA variables at the time the STORE statement is executed. In other words, prior to the execution of a statement involving set occurrence selection LOCATION MODE OF OWNER, the run-unit must initialize any data-names or identifiers specified in this phrase.
6. The LOCATION MODE OF OWNER option causes DBCS to select a particular set occurrence of set-name-1 on the basis of the LOCATION MODE clause specified in the record entry for the owner of set-name-1. There are essentially two cases:
   a. The owner record of the set occurrence can be uniquely identified on the basis of its LOCATION MODE clause alone (i.e., the record entry for the owner record contains LOCATION MODE is DIRECT or CALC).
   b. The owner record of the set occurrence to be selected cannot be determined except in terms of its membership in some other set. This will occur when the LOCATION MODE clause in the record entry for the owner is VIA set-name. In this circumstance, selection is governed by the USING option of this SET OCCURRENCE SELECTION phrase and by the SET OCCURRENCE SELECTION phrase for the "VIA set" named in the LOCATION MODE clause of the owner record. The data-items specified in the USING option must uniquely identify a specific record occurrence within an occurrence of the set named in the LOCATION MODE clause of the owner record. There is only one way in which a record occurrence can be uniquely identified in terms of its membership in a set occurrence in DBMS, and that is by its sort key(s). Thus, if the SET OCCURRENCE SELECTION is

LOCATION MODE OF OWNER and the owner's LOCATION MODE is VIA set-name-2, the owner's MEMBER clause in set-name-2 must contain the ORDER IS SORTED phrase and some form of DUPLICATES NOT ALLOWED option. In addition, each data-name specified as a sort key of that MEMBER clause must appear in a USING phrase within this SET OCCURRENCE SELECTION phrase. Additionally, the SET OCCURRENCE SELECTION phrase for set-name-2 named in the owner's LOCATION MODE clause may, in turn, specify LOCATION MODE OF OWNER and the LOCATION MODE may again be VIA set-name. This condition may occur to an arbitrary number of levels, but must eventually terminate with a SET OCCURRENCE SELECTION phrase that does not specify a LOCATION MODE OF OWNER for which the LOCATION MODE is VIA set-name. At each level other than the first, the arguments specified in a USING phrase are used to select an owner record in its capacity as a member of another set. The arguments specified must be the appropriate sort keys in the record(s) to be selected.

7. The optional ALIAS phrase provides for the situation where you have defined a given record as a member in more than one set type, and each such set type has the same owner record type, and each member clause has a SET OCCURRENCE SELECTION of LOCATION MODE OF OWNER. In this situation, more than one argument value may be required for the data-item named as an argument. The ALIAS phrase provides the UWA locations for such values.
   a. Data-name-4. . . must, if the LOCATION MODE clause in the record entry for the owner record is DIRECT or CALC, refer to data-items specified in that LOCATION MODE clause. If, however, the LOCATION MODE is VIA set-name, data-name-4. . . must refer to data-items specified in the USING phrase of a SET OCCURRENCE SELECTION phrase for another set with the same defined owner record type.
   b. By their appearance in an ALIAS phrase, all identifiers/pseudonyms are implicitly defined as having the same characteristics as their corresponding data-names. That is, an identifier specified in an ALIAS phrase defines an identifier in the run-unit's User Working Area. Pseudonym-1, if specified, must be preceded by a percent sign. It can contain up to six characters and cannot contain hyphens. You should provide it for FORTRAN use when identifier-1 contains more than six characters and/or hyphens.
   c. If a sort or CALC key consists of more than one data item, you need not specify an ALIAS for every item.
8. If you use the LOCATION MODE OF OWNER option, the record entry for the owner record type being referenced must have a DUPLICATES NOT ALLOWED phrase if its LOCATION MODE is CALC. Similarly, you would specify the DUPLICATES NOT ALLOWED phrase for the sort key specified in the USING (or ALIAS) option.
9. Where you have specified LOCATION MODE OF OWNER in a SET OCCURRENCE SELECTION phrase and the LOCATION MODE is VIA, the use of the optional word RANGE in any of the sort subkeys of the OWNER causes a value of such a subkey to represent a range of values and has the following implications for the process of set occurrence selection.
   An equality match between the range key, which is in the record to be selected, and the input argument value in the User Working Area is not required for a record to be selected as being the owner of the sought set occurrence.
   A match will occur (regardless of whether the range key has been specified as ascending or descending) under the following conditions:
   a. If the input argument value in the User Working Area equals the value of any specific range key.
   b. If the input argument value in the User Working Area is less than the value of any specific range key. A match will occur on the range key with the lowest value.
   c. If the input argument value in the User Working Area lies between two adjacent range key values. The match will occur with the larger range key value.

Example

```
SET OCCURRENCE SELECTION CURRENT.
```

CHAPTER 5

THE SUB-SCHEMA DATA DESCRIPTION LANGUAGE (DDL)

The Sub-Schema Data Description Language (DDL) enables you to describe the subset of a data base known to one or more application programs. A sub-schema description written in the Sub-Schema DDL consists of the Sub-Schema Identification and three sections:

AREA SECTION

RECORD SECTION

SET SECTION

The sections must appear in the above order; they consist of an entry for each area, record, or set to be included in the sub-schema being defined.

Sub-schema descriptions must appear after the schema description, but before the END-SCHEMA indicator. They are passed through the DDL processor, SCHEMA, along with the DMCL and Schema DDL statements.

SUB-SCHEMA ENTRY

Function

Use the sub-schema entry to define and name a sub-schema within a schema, and to specify the privacy lock for access to the sub-schema by run-units.

General Format

```
SUB-SCHEMA NAME IS sub-schema-name
    [PRIVACY LOCK IS lock-1].
```

Technical Notes

1. Sub-schema-name must be unique among the sub-schema-names at your installation.
2. Lock-1 must be a single alphanumeric name up to five characters long.
3. Up to 36 sub-schemas are allowed in a schema.

Example

```
SUB-SCHEMA NAME IS SUB01 PRIVACY SALEX.
```

SUB-SCHEMA AREA SECTION

Function

Use the area section to declare the areas of the schema that are to be included in the sub-schema and, by implication, to remove from view all other areas of the schema.

General Format

Format 1

```
AREA SECTION.
    COPY area-name-1 [area-name-2] ... .
```

Format 2

```
AREA SECTION.
    COPY ALL AREAS.
```

Format 3

```
AREA SECTION.
    COPY TEMPORARY area-name-3 [area-name-4] ... .
```

Technical Notes

1. Area-name-1, area-name-2, . . . must refer to areas defined in the schema.
2. If Format 2 is used, Formats 1 and 3 entries are not allowed. Otherwise 1 and 3 may be repeated as needed.
3. Format 1 causes the entries for the referenced areas in the schema to be included in the sub-schema.
4. Format 2 causes all areas for which entries are included in the schema to be included in the sub-schema.
5. Format 3 designates the named areas to be sub-schema temporary. A sub-schema temporary area is a private unique occurrence of a normal area. Any changes made to the area will be discarded when the run-unit closes the area or terminates. This is used to permit program testing on "live" data without loss of data base integrity.

Example

```
AREA SECTION.
    COPY MARKETING-AREA, INVENTORY-AREA.
```

SUB-SCHEMA RECORD SECTION

Function

Use the record section to define the records of the schema that are to be included in the sub-schema. By implication, you will also be removing from view all other records of the schema. However, you must be aware that when DBCS is, for example, searching for the owner record of a set and the current of a set is not linked directly to the owner, it must go through every record in the set until it reaches the owner. If DBCS finds a record during its search that is not in the sub-schema, it issues an error. Thus, you must make certain that the record linkages, at least, are present for the sets in the sub-schema.

General Format

Format 1

```
RECORD SECTION.

01  record-name-1.
01  record-name-2.
[ ||02  data-name-1.                          ||]
  ||[COPY sub-schema-name TEXT.]              ||
  [COPY OTHERS .]
```

Format 2

```
RECORD SECTION.
COPY ALL RECORDS.
```

Technical Notes

1. All record-names must be the names of records declared in the schema.
2. Each record named must be located within an area named in the Area Section. If a record type is located in more than one area, and not all of these areas are included in the sub-schema, only the record occurrences located in the included areas are accessible from the sub-schema.
3. Format 1 causes the entries for the referenced records in the schema to be included in the sub-schema.
4. Format 2 causes all records included in the areas specified in the Area Section to be included in the sub-schema.
5. If you use Format 2, you cannot use Format 1; otherwise you can repeat Format 1 as needed.
6. You can use Format 1 in the following ways.
   a. You can specify just the record-name and all of the data-items in that record will be included in the sub-schema. For example:

      ```
      01  CUSTOMER-REC.
      ```

   b. You can specify the data-items to be included in the sub-schema either explicitly or by means of the COPY OTHERS phrase. If you do not specify a data-item and you do not use the COPY OTHERS phrase, the data-item will not be included in the sub-schema. For example:

```
01  CUSTOMER-REC.
 02  CUST-NAME.
 02  CUST-ADD.
 02  CUST-ID.
 COPY OTHERS.
```

will cause all of the data-items in CUSTOMER-REC to be included in the sub-schema. However,

```
01  CUSTOMER-REC.
 02  CUST-NAME.
 02  CUST-ADD.
 02  CUST-ID.
```

will cause only the data-items specified to be included in the sub-schema.

If you do not include data-items that are CALC keys, DBCS will not be able to store the CALC record nor find it using FIND rse 5. Set occurrence selection can be affected if the CALC key is used in an ALIAS phrase or the record is the owner of a set with set occurrence selection LOCATION MODE OF OWNER. Similarly, if you do not include data-items that are sort keys, set occurrence selection can be affected, new records cannot be stored in the set, no sort data-items can be modified, and the record cannot be inserted or deleted.

c. You can specify the record-name followed by a line containing just 02 and a period (.). This causes the linkages in the record, but not the data, to be included in the sub-schema. For example:

```
01  CUSTOMER-REC.
 02.
```

d. You can specify text that will accompany a data-aggregate in the sub-schema. The text can be any information that you wish to include. It can be a 03- or greater level item (for COBOL applications), a FORTRAN declaration, or a comment. You specify the text on a line-by-line basis. Thus, any line following a 02-level data-item will be treated as text unless it begins with COPY, 02, 01, or SET. If you wish to begin a text line with one of these words, you must prefix the line with a plus sign (+). You can enter a line of null text (to override an implicit COPY TEXT) by typing a plus sign followed by a carriage-return.

When you specify text for a data-item in a sub-schema, any following sub-schemas containing that data-item will also include that text unless you change it. You can change it by including different text for that data item in one of the subsequent sub-schemas. This new text will then be used for the following sub-schemas until you change it again. You can change back to a previous text for a data-item by including the COPY sub-schema-name TEXT statement following the line containing the data-item. The text for the data-item from the named sub-schema will be used in the current sub-schema and all subsequent sub-schemas that contain that data item until you change the text again. For example:

```
RECORD NAME IS R1
        .
        .
        .
02  D1 SIZE 4 WORDS.
02  D2 TYPE FLOAT BIN.
02  D3 PIC X(5) OCCURS 4.
        .
        .
        .
```

```
SUB-SCHEMA NAME IS SS1.
          .
          .
          .
01  R1.
  02  D1.
      03  DD1 PIC X(12).
      03  DD2 PIC S9(18) USAGE COMP.
  COPY OTHERS.
          .
          .
          .
SUB-SCHEMA NAME IS SS2.
          .
          .
          .
01  R1.
  02  D1.
      EQUIVALENCE (D1(1),D6BIT),(D1(3),DPINT)
  02  D2.
          .
          .
          .
SUB-SCHEMA NAME IS SS3.
01  R1.
  02  D1.
      COPY SS1.
 COPY OTHERS.
```

Sub-schemas SS1 and SS3 have the same record descriptions and would be used for COBOL application programs. Sub-schema SS2 would be used for a FORTRAN application. If another sub-schema were declared and D1 were included, the COBOL text would be used for D1 unless you explicitly changed it.

Example

```
RECORD SECTION.
01  CUSTOMER-RECORD.
01  ORDER-RECORD.
01  STOCK-RECORD.
01  SUPPLIER-RECORD.
    02  NAME.
    02  ADDR.
    02  RATE.
```

SUB-SCHEMA SET SECTION

Function

Use the set section to define the sets of the schema that are to be included in the sub-schema and, by implication, to remove from view all other sets of the schema.

General Format

Format 1

```
SET SECTION.
    COPY set-name-1 [set-name-2] . . . .
```

Format 2

```
SET SECTION.
    COPY ALL SETS.
```

Technical Notes

1. All set names must refer to sets defined in the schema.
2. You can repeat Format 1 entries as required. If you use Format 2, no Format 1 entries are allowed.
3. Format 1 causes the entries for the referenced sets in the schema to be included in the sub-schema.
4. Format 2 causes all sets for which entries are included in the schema to be included in the set section.
5. You must include an entry in the Record Section for the owner record of each set included in a sub-schema.
6. If a set included in a sub-schema is sorted on one or more keys, you should include all the record types containing those keys in the sub-schema if you want record modification to be permitted for members of that set.

Example

```
SET SECTION.
    COPY ORDER-SET, INVENTORY-SET, SUPPLIER-SET.
```

END-SCHEMA

Function
Use the END-SCHEMA statement to end your DDL file.

General Format

END-SCHEMA .

Technical Notes

1. You must include the descriptions of all sub-schemas before the END-SCHEMA statement.

5.1 EXAMPLE OF A SCHEMA WITH SUB-SCHEMAS

Consider a research company that is organized into departments. Some of these departments are actively engaged in research; others are support departments (e.g., accounting, payroll, legal, etc.). Each research department is broken up into research project groups, although there are some special projects that overlap departments. The research projects are supported by contracts and grants that come from external sources – some projects receiving funds from more than one contract or grant. The employees of the company are grouped according to department, project, and personnel classification. Research personnel (other than project leaders) work only at one project at a time. There are, though, some project leaders who administer more than one project. The following example shows how to create a data base that contains the data and the data relationships that describe this activity. Figure 5-1 shows the set relationships defined in the example schema and sub-schema illustrated in Figure 5-2.

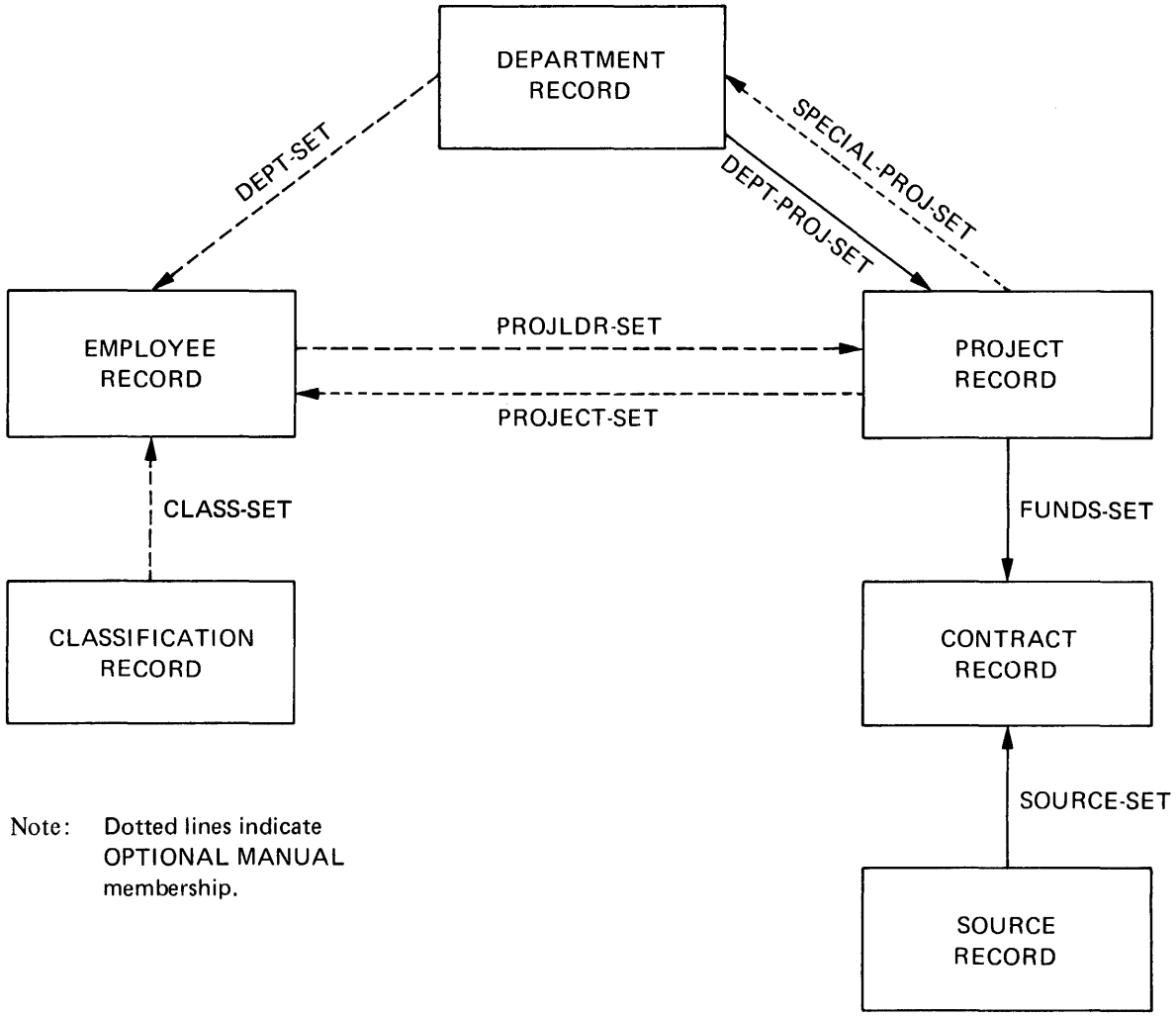


Figure 5-1 Set Relationships in Example Schema

```
00100    IMAGES BY COMMAND.
00200    NOTE ALL.
00300    INTERCEPT BIND.
00400    JOURNAL JRN1.
00500    RECORDS-PER-PAGE 35.
00600
00700    ASSIGN PERSONNEL-AREA TO FILE01
00800    BACKUP BEFORE IMAGES
00900    BUFFER 4
01000    CALC 2 RPP
01100    FIRST PAGE IS 1
01200    LAST PAGE IS 100
01300    PAGE SIZE IS 512 WORDS
01400    RANGE OF CLASSIFICATION-RECORD IS 2 TO 50.
01500
01600    ASSIGN RESEARCH-AREA TO FILE02
01700    BACKUP BEFORE IMAGES
01800    BUFFER 4
01900    CALC 2 RPP
02000    FIRST PAGE IS 300
02100    LAST PAGE IS 360
02200    PAGE SIZE IS 512 WORDS
02300    RANGE OF PROJECT-RECORD IS 300 TO 340.
02400
02500    SCHEMA NAME IS SCHEX.
02600
02700    AREA NAME IS PERSONNEL-AREA
02800             PRIVACY EXCLUSIVE UPDATE IS PREXP
02900             PRIVACY PROTECTED UPDATE IS PRPTP
03000             PRIVACY RETRIEVAL IS PRTLK.
03100
03200    AREA NAME IS RESEARCH-AREA
03300             PRIVACY LOCK RSCLK.
03400
03500    RECORD NAME IS DEPARTMENT-RECORD
03600             LOCATION MODE IS DIRECT DPTKEY
03700             WITHIN PERSONNEL-AREA.
03800
03900             02 DEPARTMENT PIC X(20).
04000
04100    RECORD NAME IS EMPLOYEE-RECORD
04200             LOCATION MODE VIA CLASS-SET
04300             WITHIN PERSONNEL-AREA.
04400
04500             02 EMPLOYEE PIC X(25).
04600             02 HOME-ADDRESS PIC X(40).
04700             02 HOME-PHONE PIC 9(10).
04800             02 SS-NUMBER PIC 9(9).
04900             02 MARITAL-STATUS PIC X.
05000             02 HIRED PIC 9(8).
05100             02 DEPENDENTS PIC 99.
05200             02 BASE-SALARY TYPE FIXED BINARY REAL 15,2.
```

Figure 5-2 Schema/Sub-schema Example

```
05300
05400    RECORD NAME IS CLASSIFICATION-RECORD
05500             LOCATION MODE IS CALC USING CLASSIFICATION
05600             DUPLICATES ARE NOT ALLOWED
05700             WITHIN PERSONNEL-AREA.
05800
05900             02 CLASSIFICATION PIC X(15).
06000             02 LEVEL PIC 99.
06100
06200    RECORD NAME IS PROJECT-RECORD
06300             LOCATION MODE CALC USING PROJECT-ID
06400             DUPLICATES ARE NOT ALLOWED
06500             WITHIN RESEARCH-AREA.
06600
06700             02 PROJECT-ID PIC 99V9999.
06800             02 PROJECT-TITLE PIC X(30).
06900
07000    RECORD NAME IS CONTRACT-RECORD
07100             LOCATION MODE VIA FUNDS-SET
07200             WITHIN RESEARCH-AREA.
07300
07400             02 CONTRACT-NUMBER PIC 9(7).
07500             02 CONTRACT-DATE PIC 9(10).
07600             02 COMPLETION-DATE PIC 9(10).
07700             02 INITIAL-AMOUNT TYPE FIXED 21,2.
07800             02 BALANCE TYPE FIXED 21,2.
07900
08000    RECORD NAME IS SOURCE-RECORD
08100             LOCATION DIRECT SRCKEY
08200             WITHIN RESEARCH-AREA.
08300
08400             02 SOURCE-NAME PIC X(25).
08500             02 SOURCE-ADDRESS PIC X(40).
08600             02 SOURCE-PHONE PIC 9(10).
08700             02 SOURCE-CONTACT PIC X(25).
08800             02 SOURCE-TITLE PIC X(10).
08900
09000    SET NAME IS DEPT-SET
09100             MODE IS CHAIN LINKED TO PRIOR
09200             ORDER IS SORTED DUPLICATES NOT ALLOWED
09300             OWNER IS DEPARTMENT-RECORD
09400             MEMBER IS EMPLOYEE-RECORD OPTIONAL MANUAL
09500                      LINKED TO OWNER
09600                      ASCENDING KEY IS SS-NUMBER
09700             SET SELECTION CURRENT.
09800
```

Figure 5-2 (Cont.) Schema/Sub-schema Example

```
09900    SET NAME IS CLASS-SET
10000            MODE IS CHAIN LINKED TO PRIOR
10100            ORDER IS SORTED DUPLICATES
10200            OWNER IS CLASSIFICATION-RECORD
10300            MEMBER IS EMPLOYEE-RECORD OPTIONAL MANUAL
10400                     LINKED TO OWNER
10500                     ASCENDING KEY IS BASE-SALARY
10600            SET SELECTION CURRENT.
10700
10800    SET NAME IS PROJECT-SET
10900            MODE IS CHAIN LINKED TO PRIOR
11000            ORDER IS ALWAYS LAST
11100            OWNER IS PROJECT-RECORD
11200            MEMBER IS EMPLOYEE-RECORD OPTIONAL MANUAL
11300                     LINKED TO OWNER
11400            SET SELECTION CURRENT.
11500
11600    SET NAME IS PROJLDR-SET
11700            MODE IS CHAIN LINKED TO PRIOR
11800            ORDER IS ALWAYS NEXT
11900            OWNER IS EMPLOYEE-RECORD
12000            MEMBER IS PROJECT-RECORD OPTIONAL MANUAL
12100                     LINKED TO OWNER
12200            SET SELECTION CURRENT.
12300
12400    SET NAME IS DEPT-PROJ-SET
12500            MODE IS CHAIN LINKED TO PRIOR
12600            ORDER IS SORTED DUPLICATES
12700            OWNER IS DEPARTMENT-RECORD
12800            MEMBER IS PROJECT-RECORD MAND AUTO
12900                     LINKED TO OWNER
13000                     ASCENDING KEY IS PROJECT-ID
13100            SET SELECTION CURRENT.
13200
13300    SET NAME IS SPECIAL-PROJ-SET
13400            MODE IS CHAIN LINKED TO PRIOR
13500            ORDER IS ALWAYS LAST
13600            OWNER IS PROJECT-RECORD
13700            MEMBER IS DEPARTMENT-RECORD OPTIONAL MANUAL
13800                     LINKED TO OWNER
13900            SET SELECTION CURRENT.
14000
14100    SET NAME IS FUNDS-SET
14200            MODE IS CHAIN LINKED TO PRIOR
14300            ORDER IS ALWAYS LAST
14400            OWNER IS PROJECT-RECORD
14500            MEMBER IS CONTRACT-RECORD MAND AUTO
14600                     LINKED TO OWNER
14700            SET SELECTION CURRENT.
14800
```

Figure 5-2 (Cont.) Schema/Sub-schema Example

```
14900    SET NAME IS SOURCE-SET
15000            MODE IS CHAIN LINKED TO PRIOR
15100            ORDER IS ALWAYS LAST
15200            OWNER IS SOURCE-RECORD
15300            MEMBER IS CONTRACT-RECORD MAND AUTO
15400                     LINKED TO OWNER
15500            SET SELECTION CURRENT.
15600
15700    SUB-SCHEMA NAME IS FUNDING
15800            PRIVACY LOCK IS DNUF.
15900    AREA SECTION.
16000            COPY RESEARCH-AREA.
16100    RECORD SECTION.
16200            01 SOURCE-RECORD.
16300            01 CONTRACT-RECORD.
16400            01 PROJECT-RECORD.
16500    SET SECTION.
16600            COPY SOURCE-SET, FUNDS-SET.
16700
16800    SUB-SCHEMA NAME IS DEPT-PROJ
16900            PRIVACY LOCK IS DEPRJ.
17000    AREA SECTION.
17100    COPY TEMPORARY PERSONNEL-AREA, RESEARCH-AREA.
17200    RECORD SECTION.
17300            01 DEPARTMENT-RECORD.
17400            01 PROJECT-RECORD.
17500            01 CONTRACT-RECORD.
17600    SET SECTION.
17700            COPY DEPT-PROJ-SET, SPECIAL-PROJ-SET, FUNDS-SET.
17800
17900    SUB-SCHEMA NAME IS TOTAL
18000            PRIVACY LOCK ETHNG.
18100    AREA SECTION.
18200            COPY ALL AREAS.
18300    RECORD SECTION.
18400            COPY ALL RECORDS.
18500    SET SECTION.
18600            COPY ALL SETS.
18700
18800    END-SCHEMA.
```

Figure 5-2 (Cont.) Schema/Sub-schema Example

CHAPTER 6
DBMS UTILITIES

DBMS includes utility routines that can help you maintain the data base. They are:

DBINFO – a program that gives information about the data base and the schema.
DBMEND – a program that uses the journal file to perform page recovery.
DAEMDB – a program that copies the journal file onto magnetic tape.

6.1 THE DBINFO PROGRAM

DBINFO is a utility program that you can use to obtain information about the data base such as:

- a cross-reference listing of all symbolic names in a sub-schema.
- a map of each record type in a sub-schema.
- the usage of record occurrences on a page or set basis.
- the actual data values of the record occurrences.
- the amount of free space on pages of the data base.

6.1.1 Using DBINFO

DBINFO works with one sub-schema at a time. For example, if you open all areas, you will open only those areas in that sub-schema. If you want information about the entire data base, you can equate the schema to a sub-schema by using a variation of the SS command.

The DISPLAY command is the means by which you can specify exactly what kind of information you want from DBINFO. If you want to process the data base as opposed to the schema (e.g., find out the amount of free space), you must specify the scope of what DBINFO will display by specifying the range (or ranges) of pages that you desire to process. If you do not explicitly give any page ranges, DBINFO will give you information from all pages in the open areas.

When you run DBINFO, you must give the specification of the file into which DBINFO will put the information you request. If you specify an existing file, DBINFO will append to the file (if you use the APPEND command), or supersede the file (if you use the SUPERSEDE command). If the file does not exist, DBINFO will create it, whether you use the APPEND or SUPERSEDE command.

You can use the output file for more than one report from the same or different sub-schemas. You must be sure to open the relevant areas for the particular sub-schema you are processing.

If you wish to write more than one information file during the running of DBINFO, just open each file by means of the APPEND or SUPERSEDE command.

6.1.2 DBINFO Commands

The commands to DBINFO are described on the following pages in alphabetical order, each command starting a new page. However, the command line format is described before the commands because it applies to all of them. DBINFO uses the SCAN program to process the command lines; the following rules conform to SCAN requirements.

A keyword can be arbitrarily abbreviated as long as no other keyword within the same context has the same abbreviation. The context of a keyword is the set of all keywords that can appear in the same place. Thus, all commands are in one context, and the set of possible arguments to each command is each in a separate context. For example, you must spell out ALL when you use it as the argument to the CLOSE command because you could use an area-name in the

same context. On the other hand, you can use DISPLAY C instead of DISPLAY CREF, even though C is the abbreviation for the CLOSE command, because C is in the context of an argument to the DISPLAY command. The allowable minimum abbreviations for command are shown in Table 6-1.

Table 6-1
DBINFO Commands and Abbreviations

Command	Abbreviation
APPEND	A
CLOSE	C
DISPLAY	D
OPEN	O
PAGES	P
SCHEMA	SC
SS	SS
SUPERSEDE	SU

Spaces and tabs are ignored in a command line; and you must separate items in lists (e.g., a list of area-names) with commas. You can use either upper or lower case in a command line. However, DBINFO will treat a value as all upper case in a comparison. For example, if you give an area-name in lower case, DBINFO will use its upper-case equivalent when checking the name against the area-name in the schema. All commands and their modifiers are shown in upper case in this chapter.

DBINFO will accept only names composed of alphanumeric characters. Thus, if you use a special character such as hyphen (-), you must enclose the entire name in double quotation marks so that DBINFO will treat it as a name. For example, CLOSE A-B will not be accepted, but CLOSE "A-B" will.

You can continue a command on another line by placing a hyphen as the last (rightmost) character on the line before the carriage return. The next line is then considered to be a continuation of the hyphen-ended line.

DBINFO will accept indirect command files. That is, you can store one or more commands in a file and then submit the file to DBINFO by putting its file specification preceded by an at sign (@) on the command line.

```
.R DBINFO
/@file specification
```

The file specification is the device, filename, extension, and project-programmer number of the indirect file.

You can exit from DBINFO by typing CTRL/C(^C).

APPEND

Use the APPEND command to give the file specification of the output file. This file will contain the information that you request from DBINFO. If there is already a file containing information, DBINFO will append the new information to it. If there is no existing file, DBINFO will create one. The form of the APPEND command is:

APPEND file specification

The file specification is of the form:

dev:file.extension

where:

dev:	is a device-name. If you do not specify a device, DBINFO will use DSK:.
file	is the name of the file. You must specify the name if the device is DSK:.
extension	is the extension for the file. If you do not specify an extension, DBINFO will use .DBI.

Example

/APPEND INFO.DBI

CLOSE

Use the CLOSE command to close one or more data-base areas that you have previously opened. The formats of the CLOSE command are:

1. CLOSE area-name-1 [,area-name-2] . . .
2. CLOSE ALL

The first format causes DBINFO to close the named data-base area or areas. The second format causes DBINFO to close all open data-base areas within the current sub-schema.

The CLOSE command does not affect the information file, only data-base areas.

Example

```
/CLOSE AREA1, PERSONNEL
```

DISPLAY

Use the DISPLAY command to specify to DBINFO the kind of information you want about the data base or the schema. You can give only one keyword argument at a time to the DISPLAY command. But you can use the DISPLAY command as many times as you wish during the running of DBINFO. DBINFO will just append the requested report to the current information file. In this way, you can get all of the possible forms of information about the data base in the same information file.

The formats of the DISPLAY command are:

1. DISPLAY CREF
2. DISPLAY MAP
3. DISPLAY USAGE [:set-name]
4. DISPLAY DATA [:set-name]
5. DISPLAY FREE [:integer]

DISPLAY CREF causes DBINFO to output an alphabetical listing of each symbolic name in a sub-schema. With each name, DBINFO includes its kind (e.g., record-name), how it is used (e.g., owner), and where it is used (e.g., the set in which it is the owner).

DISPLAY MAP causes DBINFO to output a map containing each record type in the sub-schema. Included in the map are the record type IDs of the records, their pointers, and data-fields.

DISPLAY USAGE causes DBINFO to output the usage of the record types. If you do not include an argument, DBINFO describes the number of record occurrences for each record type on each object page (see the PAGE command). If a page is empty, DBINFO does not include any information. If you give a set-name as an argument, DBINFO gives you the pages on which the occurrences of the set's owner are located and the number of members that are on each owner's page. It also gives the total number of occurrences of the set and the total number of member records.

DISPLAY DATA causes DBINFO to output a symbolic dump of the actual values in each record. If you do not specify an argument, DBINFO dumps each record occurrence on each object page. If a page is empty, DBINFO does not include any information. If you specify a set-name as an argument, DBINFO dumps each record occurrence in order according to its place in each object set occurrence. DBINFO processes a particular set occurrence if its owner record is on an object page.

DISPLAY FREE causes DBINFO to output a list of free data space per page. If you specify an argument, DBINFO uses the integer to determine what constitutes a full page. Any page that has fewer free words than the integer you specified is considered full and will not be included in the list. If you do not specify an argument, DBINFO uses an argument of 0. For the pages listed, DBINFO gives the page number, the number of free words, and asterisks that each signify ten free words. If there are at least three totally empty or full pages in a row, DBINFO will output the range of their page numbers and specify whether they are empty or full. Otherwise, those pages will not be included in the list.

Example

```
/DISPLAY CREF
```

OPEN

Use the OPEN command to open one or more data-base areas in PROTECTED RETRIEVAL mode. The formats of the OPEN command are:

1. OPEN area-name-1 [,area-name-2] . . . [:key-1]
2. OPEN ALL [:key-1]

The first format causes DBINFO to open the specified area or areas in PROTECTED RETRIEVAL mode. You can specify up to eight areas in one command. The second format causes DBINFO to open all areas of the sub-schema currently being used. In either form of the command, you must specify the privacy key for PROTECTED RETRIEVAL (key-1) if the areas have privacy locks. The areas must all have the same privacy lock (or none) if you specify them together in one OPEN command. If the areas that you wish to use have different privacy locks, you will have to use separate OPEN commands to open them.

Example

```
/OPEN AREA1, PERSONNEL:PRETV
```

PAGES

Use the PAGES command to restrict the pages (i.e., the object pages) of the data base that DBINFO will use when applying the DISPLAY command. That is, DBINFO will only include the specified pages when it outputs the information requested in a DISPLAY command. If you do not give a PAGES command before a DISPLAY command, DBINFO will use all pages of all open areas.

The formats of the PAGES command are:

1. PAGES

2. PAGES ‖ integer
integer-integer
area-name ‖

The first form of the PAGES command causes DBINFO to use all pages in all open areas.

The second form of the PAGES command allows you to specify up to 16 page ranges separated by commas. If you specify an integer, the range is only that page. If you specify two integers separated by a hyphen, you specify all the pages from the first integer through the second (e.g., 13-25 includes pages 13 through 25). If you include an area-name, you include all the pages in that area.

Example

```
/PAGES 4,7,13-25, "CUST-AREA"
```

SS

Use the SS command to specify the sub-schema to be used by DBINFO. This means that DBINFO will be aware of only those areas, record-types, and data-items included in the specified sub-schema. You must issue an SS command before issuing your first DISPLAY command so that DBINFO will list the information for the sub-schema you want.

The format of the SS command is:

```
SS [sub-schema-name]
```

If you do not specify an argument, DBINFO will use the entire schema as the sub-schema. If you specify a sub-schema-name, DBINFO will use the specified sub-schema. You can only specify up to eight sub-schemas during one run of DBINFO.

Example

```
/SS SUBSCH1
```

SUPERSEDE

Use the SUPERSEDE command to give the file specification of the output file. This file will contain the information that you request from DBINFO. If there is already a file of that name containing information, DBINFO will supersede that information. If there is no existing file, DBINFO will create one.

The form of the SUPERSEDE command is:

SUPERSEDE file specification

The file specification is of the form:

dev:file.extension

where:

dev:	is a device-name. If you do not specify a device, DBINFO will use DSK:.
file	is the name of the file. You must specify the name if the device is DSK:.
extension	is the extension for the file. If you do not specify an extension, DBINFO will use .DBI.

Example

/SUPERSEDE DBINF.SS1

SCHEMA

Use the SCHEMA command to identify to DBINFO the schema that you wish to use. The format of the SCHEMA command is:

SCHEMA schema-name

Schema-name is the name of an existing schema.

Example

/SCHEMA SCHEX

6.1.3 DBINFO Error Messages

%INFAAD AREA STATUS ALREADY DESIGNATED FOR area-name

You tried to open an area already open. Close the area and try again.

?INFAAO AREA area-name NOT IN "READY" STATE -- SEE FORCEOPEN IN DBMEND MANUAL

The area cannot be opened because its Area Status Record is bad. Use the DBMEND FORCEOPEN command (Section 6.2.3) to adjust the Area Status Record and try DBINFO again.

?INFACS AREA area-name IN CREATION STATE -- SEE EXCEPTION 43 IN MANUAL

The area is currently being used. See Appendix B in the *DBMS Programmer's Procedures Manual* for exception code 43.

?INFANO PAGE TO DISPLAY IN CLOSED AREA area-name

The page you specified is in an area that is not open. Open the area or specify another page.

?INFAOE ATTEMPT TO OPEN EMPTY AREA . . . FOR RETRIEVAL

You tried to open an empty area. Put data into the area before you use DBINFO on it.

?INFCDP CAN'T DEFAULT PAGE TUPLES -- NO AREAS OPEN

DBINFO cannot use a default page range because no areas are open. Open at least one area and try again.

?INFDAC DATA CHANNELS ALL IN USE

DBINFO cannot open all files specified because not enough JFNs are available. Try again with fewer areas.

?INFDAF DATA BASE ACCESS FAILURE -- IS DBMS PROBLEM IF IT RECURS

DBINFO could not access the data base. Check that the DBS files are in an accessible directory. If the error recurs, submit an SPR.

?INFIAN INVALID AREA NAME area-name

You specified an invalid area-name. Try again with a valid area-name.

?INFNNA page-range NOT NUMERIC OR AN AREA-NAME

You specified a page-range that was neither numeric nor an area-name. Try again with a correct page-range.

?INFNOS NO OUTPUT FILE HAS BEEN SUPPLIED AS YET

You tried to get a report before specifying an output file. Use the APPEND or SUPERSEDE command to specify a file.

?INFNSK NO SCHEMA NAME HAS BEEN SUPPLIED AS YET

You tried to get a report before specifying the schema. Use the SCHEMA command to specify the schema.

?INFNSS NO SUB-SCHEMA NAME HAS BEEN SUPPLIED AS YET

You tried to get a report before specifying a sub-schema. Use the SS command to specify a sub-schema (or to specify that the schema will be used as the sub-schema).

?INFPKI PRIVACY KEY INCORRECT FOR area-name

You specified an incorrect privacy key for an area. Determine the correct privacy key and try again.

?INFSSN SUB-SCHEMA NAME IS NOT IN SPECIFIED SCHEMA

You specified a sub-schema that is not in the schema you specified. Determine the correct sub-schema and try again.

?INFTMA TOO MANY AREAS ON ONE COMMAND LINE

You specified more than eight areas in one OPEN command. Try again with fewer than eight areas.

?INFTMS TOO MANY SUB-SCHEMAS ACCESSED

You specified more than eight sub-schemas for this run of DBINFO. Return to the monitor (Control C); type R DBINFO; and continue as before.

?INFTMT TOO MANY PAGE TUPLES SPECIFIED/IMPLIED

You specified (or implied if using the default) more than 16 page-ranges in one PAGE command. Try again with fewer than 16 page-ranges.

?INFTNS UNSUPPORTED DATA TYPE ENCOUNTERED

DBINFO found a data type that DBMS does not support. This is either an error in the schema or a software error. Correct the schema or submit an SPR.

?INFWCP WILDCARDING IS PROHIBITED

DBINFO does not accept wildcard format in its commands.

6.1.4 Sample DBINFO Output

The following pages contain samples of the output from the DBINFO program. They show the following:

1. Symbolic cross reference
2. Record-occurrence maps
3. Summary of free-space per page
4. Record usage per page
5. Set usage per page
6. Page data dump
7. Set occurrence data dump

You could use the following sequence of commands to DBINFO to obtain similar output.

```
.R DBINFO
/SCHEMA ORDENT
/SS SUBS1
/SUPERSEDE EXAMP
/OPEN AREA1
/DISPLAY CREF
/DISPLAY MAP
/DISPLAY FREE:10
/DISPLAY USAGE
/DISPLAY USAGE:"SLSCUS-SET"
/DISPLAY DATA
/DISPLAY DATA:"SLSCUS-SET"
^C
```

```
DBINFO report:  Symbolic Cross Reference
Context:        Schema name     ORDENT
                  its edit      74
                  its run ID    151
                Sub-schema      SUBS1

SYMBOL                          TYPE            HOW USED        USED WITH
------------------------------------------------------------------------
ALLORD-SET                      SET NAME
AREA1                           AREA NAME       WITHIN-phrase   SLSENG
                                                WITHIN-phrase   CUSTOM
AREA2                           AREA NAME       WITHIN-phrase   ORDSUM
                                                WITHIN-phrase   ITEM
                                                WITHIN-phrase   PURORD
                                                WITHIN-phrase   CUSTOM
AREA3                           AREA NAME       WITHIN-phrase   PROD
CUSORD-SET                      SET NAME
CUST-ADDRESS                    DATA NAME       COMPONENT       CUSTOM
CUST-KEY                        DATA NAME       COMPONENT       CUSTOM
CUST-NAME                       DATA NAME       CALC KEY        CUSTOM
CUSTOM                          RECORD NAME     MEMBER          SLSCUS-SET
                                                OWNER           CUSORD-SET
IDAREA                          REFERENCE NAME  AREA ID         CUSTOM
ITEM                            RECORD NAME     MEMBER          ORDITM-SET
                                                MEMBER          PROD-ITEM-SET
ITEM-LINE                       DATA NAME       SORT KEY        ORDITM-SET
                                                COMPONENT       ITEM
ITEM-NET                        DATA NAME       COMPONENT       ITEM
ITEM-PROD-NO                    DATA NAME       COMPONENT       ITEM
ITEM-QTY                        DATA NAME       COMPONENT       ITEM
ORDER-DATE                      DATA NAME       COMPONENT       PURORD
ORDER-LINES                     DATA NAME       COMPONENT       PURORD
ORDER-NET                       DATA NAME       COMPONENT       PURORD
ORDITM-SET                      SET NAME        VIA SET         ITEM
ORDNUM                          DATA NAME       CALC KEY        PURORD
ORDSUM                          RECORD NAME     OWNER           ALLORD-SET
ORDSUM-KEY                      REFERENCE NAME  DIRECT KEY      ORDSUM
ORDSUM-NO                       DATA NAME       COMPONENT       ORDSUM
ORDSUM-ORDERS                   DATA NAME       COMPONENT       ORDSUM
PROD                            RECORD NAME     OWNER           PROD-ITEM-SET
PROD-DESC                       DATA NAME       COMPONENT       PROD
PROD-IN-PROC                    DATA NAME       COMPONENT       PROD
PROD-INSTALLED                  DATA NAME       COMPONENT       PROD
PROD-ITEM-SET                   SET NAME
PROD-LEAD-TIME                  DATA NAME       COMPONENT       PROD
PROD-NO                         DATA NAME       CALC KEY        PROD
PROD-ON-HAND                    DATA NAME       COMPONENT       PROD
PROD-ON-ORDER                   DATA NAME       COMPONENT       PROD
PROD-PRICE                      DATA NAME       COMPONENT       PROD
PURORD                          RECORD NAME     MEMBER          CUSORD-SET
                                                MEMBER          ALLORD-SET
                                                OWNER           ORDITM-SET
SLSCUS-SET                      SET NAME
SLSENG                          RECORD NAME     OWNER           SLSCUS-SET
SLSENG-NAME                     DATA NAME       CALC KEY        SLSENG
SLSENG-OFFICE                   DATA NAME       COMPONENT       SLSENG
SLSENG-PHONE                    DATA NAME       COMPONENT       SLSENG
SYSTEM                          RECORD NAME
```

```
DBINFO report:  Record-occurence Maps
Context:        Schema name      ORDENT
                  its edit       74
                  its run ID     151
                Sub-schema       SUBS1


SYSTEM (TYPE ID=32)


CUSTOM (TYPE ID=33)
0/       CALC CHAIN
1/       NEXT  of        CUSORD-SET(O)
2/       PRIOR of        CUSORD-SET(O)
3/       NEXT  of        SLSCUS-SET(M)
4/       ******          1: CUST-NAME
5/       ******
6/       ******
7/       ******
8/       ******
9/       *               2: CUST-KEY
10/      ******          3: CUST-ADDRESS
11/      ******
12/      ******
13/      ******
14/      ******
15/      ******
16/      ******
17/      ******
18/      ******
19/      ******
20/      ******
21/      ******
22/      ******
23/      ******
24/      ******

PURORD (TYPE ID=34)
0/       CALC CHAIN
1/       NEXT  of        CUSORD-SET(M)
2/       PRIOR of        CUSORD-SET(M)
3/       OWNER of        CUSORD-SET
4/       NEXT  of        ORDITM-SET(O)
5/       PRIOP of        ORDITM-SET(O)
6/       INDEX-TREE of   ORDITM-SET
7/       NEXT  of        ALLORD-SET(M)
8/       ******          1: ORDNUM
9/       ******          2: ORDER-DATE
10/      ***             3: ORDER-LINES
            ***          4: ORDER-NET
11/      ******

ITEM (TYPE ID=35)
0/       NEXT  of        ORDITM-SET(M)
1/       PRIOR of        ORDITM-SET(M)
2/       OWNER of        ORDITM-SET
3/       NEXT  of        PROD-ITEM-SET(M)
4/       ******          1: ITEM-LINE
5/       ***
            ***          2: ITEM-PROD-NO
```

Pointers

Data Fields

Used as ID in a dump.

Bytes in Field

1 Asterisk = 1 Byte

```
6/       *****
              *          3: ITEM-QTY
7/       *
          *****          4: ITEM-NET
8/       ****

PROD (TYPE ID=36)
0/       CALC CHAIN
1/       NEXT  of        PROD-ITEM-SET(O)
2/       ******          1: PROD-NO
3/       **
           ****          2: PROD-DESC
4/       ******
5/       ******
6/       ******
7/       ******
8/       ******
9/       ******
10/      ******
11/      **
           ****          3: PROD-PRICE
12/      ****
              **         4: PROD-LEAD-TIME
13/      *
          *****          5: PROD-ON-HAND
14/      *
          *****          6: PROD-IN-PROC
15/      *****          7: PROD-ON-ORDER
              *          8: PROD-INSTALLED
16/      ****

ORDSUM (TYPE ID=37)
0/       NEXT  of        ALLORD-SET(O)
1/       PRIOR of        ALLORD-SET(O)
2/       *               1: ORDSUM-ORDERS
3/       ******          2: ORDSUM-NO

SLSENG (TYPE ID=38)
0/       CALC CHAIN
1/       NEXT  of        SLSCUS-SET(O)
2/       ******          1: SLSENG-NAME
3/       ******
4/       ******
5/       ******
6/       ******
7/       ******          2: SLSENG-OFFICE
8/       ******
9/       ******
10/      ******
11/      ******          3: SLSENG-PHONE
12/      ******
13/      ***
```

Explicitly shows starting and ending in the middle of a word.

```
DBINFO report:  Summary of Free-space/Page
Context:        Schema name      ORDENT
                  its edit       74
                  its run ID     151
                Sub-schema       SUBS1

Object page(s): 1 - 20                          <-- Argument Given in Command

PAGE NUMBER     FREE WORDS(GTR 10)      ONE ASTERISK(*)=10 FREE WORDS
   1               224                  **********************
   4               238                  ***********************
   7               212                  *********************
 8 - 10                                 (EMPTY PAGES)
   11              238                  ***********************
   14              212                  *********************
   16              227                  **********************
 17 - 20                                (EMPTY PAGES)

# OF EMPTY PAGES:   14
# OF FULL PAGES:    0
```

```
DBINFO report:  Record-usage/Page
Context:        Schema name      ORDENT
                  its edit       74
                  its run ID     151
                Sub-schema       SUBS1

Object page(s): 1 - 20


Page    1                  AREA1
1        of type           CUSTOM

Page    4                  AREA1
1        of type           SLSENG

Page    7                  AREA1
1        of type           CUSTOM
1        of type           SLSENG

Page    11                 AREA1
1        of type           SLSENG

Page    14                 AREA1
1        of type           CUSTOM
1        of type           SLSENG

Page    16                 AREA1
1        of type           CUSTOM


(TOTALS)
4        of type           CUSTOM
4        of type           SLSENG
```

```
DBINFO report:  Set-usage/Page for SLSCUS-SET
Context:        Schema name      ORDENT
                  its edit       74
                  its run ID     151
                Sub-schema       SUBS1

Object page(s): 1 - 20

OWNER LINE      MEMBERS ON PAGE(# IN SET OCCURRENCE)
   4/001                    0 (of 1)

   7/001                    1 (of 1)

   11/001                   0 (of 1)

   14/001                   0 (of 0)


# OF SET OCCURRENCES:       4
# OF MEMBER RECORDS:        1 (of 3)
```

```
DBINFO report:  Data  Dump
Context:        Schema name      ORDENT
                  its edit       74
                  its run ID     151
                Sub-schema       SUBS1

Object page(s): 1 - 20
```

```
Page   1                                                    AREA1
Words free:          224
Calc chains:       1/002

Line 1/002:        at +5 uses 25 words.                     CUSTOM
References:        1/000          107/001        107/001        11/001
1:      COPLEY
2:      8193
3:      2525 PACIFIC BEACH BLVD.          SAN DIEGO                    CALIF.       07111
```

The map ID.

```
Page   4                                                    AREA1
Words free:          238
Calc chains:        4/001

Line 4/001:         at +3 uses 14 words.                    SLSENG
References:         4/000            14/002
1:      BRANDT, D.
2:      CAMBRIDGE
3:      617491-61302522
```

```
Page   7                                                AREA1
Words free:          212
Calc chains:       7/002

Line 7/001:        at +3 uses 14 words.                 SLSENG
References:        7/000          7/002
1:      CUTHBERTSON, F.
2:      CAMBRIDGE
3:      617491-61302522

Line 7/002:        at +17 uses 25 words.                CUSTOM
References:        7/001          7/002          7/002          7/001
1:      SPAN
2:      7170
3:      25 TURNPIKE ROAD                  PROVIDENCE                    RHODE ISLAND  01800
```

```
Page   11                                                    AREA1
Words free:          238
Calc chains:       11/001

Line 11/001:       at +3 uses 14 words.                      SLSENG
References:        11/000          1/002
1:      CARMICHAEL, R.
2:      SANTA ANNA
3:      714979-24602
```

```
Page   14                                                AREA1
Words free:          212
Calc chains:       14/002

Line 14/001:       at +3 uses 14 words.                  SLSENG
References:        14/000          14/001
1:      HOGAN, R.
2:      MAYNARD
3:      617897-51112968

Line 14/002:       at +17 uses 25 words.                 CUSTOM
References:        14/001          105/001         105/001         4/001
1:      FIRST CHURCH
2:      514
3:      25 HUNTINGTON AVE.              BOSTON                          MASS.       02139
```

```
Page   16                                                AREA1
Words free:          227
Calc chains:       16/001

Line 16/001:       at +3 uses 25 words.                  CUSTOM
References:        16/000          106/001         101/001         0/000
1:      DEC
2:      0
3:      146 MAIN ST.                        MAYNARD                              MASS.         01754
```

```
DBINFO report:  Data Dump for SLSCUS-SET
Context:        Schema name      ORDENT
                  its edit       74
                  its run ID     151
                Sub-schema       SUBS1

Object page(s): 1 - 20
```

```
Page   4                                                    AREA1
Words free:          238
Calc chains:       4/001

Line 4/001:        at +3 uses 14 words.                     SLSENG
References:        4/000          14/002
1:      BRANDT, D.
2:      CAMBRIDGE
3:      617491-61302522

Line 14/002:       at +17 uses 25 words.                    CUSTOM
References:        14/001         105/001        105/001        4/001
1:      FIRST CHURCH
2:      514
3:      25 HUNTINGTON AVE.              BOSTON                          MASS.       02139
```

6.2 THE DBMEND PROGRAM AND JOURNAL USAGE

The DBMEND utility program allows you to perform page recovery on a data base. Page recovery refers to the backup and restoration of individual pages in the data base, rather than backup and recovery of entire areas (i.e., DBS files).

To perform page recovery on a data base, the DBMEND program must have a journal containing all page changes in the form of images before each change (BEFORE images) and/or after each change (AFTER images). A journal is created by a running application program as it changes the data base. Figure 6-1 shows the generation and usage of a journal file.

You can also use DBMEND to gain an abstract of the journal file or adjust the Area Status Record of the DBS file. This may be necessary if a run-unit terminates abnormally.

The journal file, the primary functions performed by DBMEND, the DBMEND commands, and DBMEND error messages are described on the following pages.

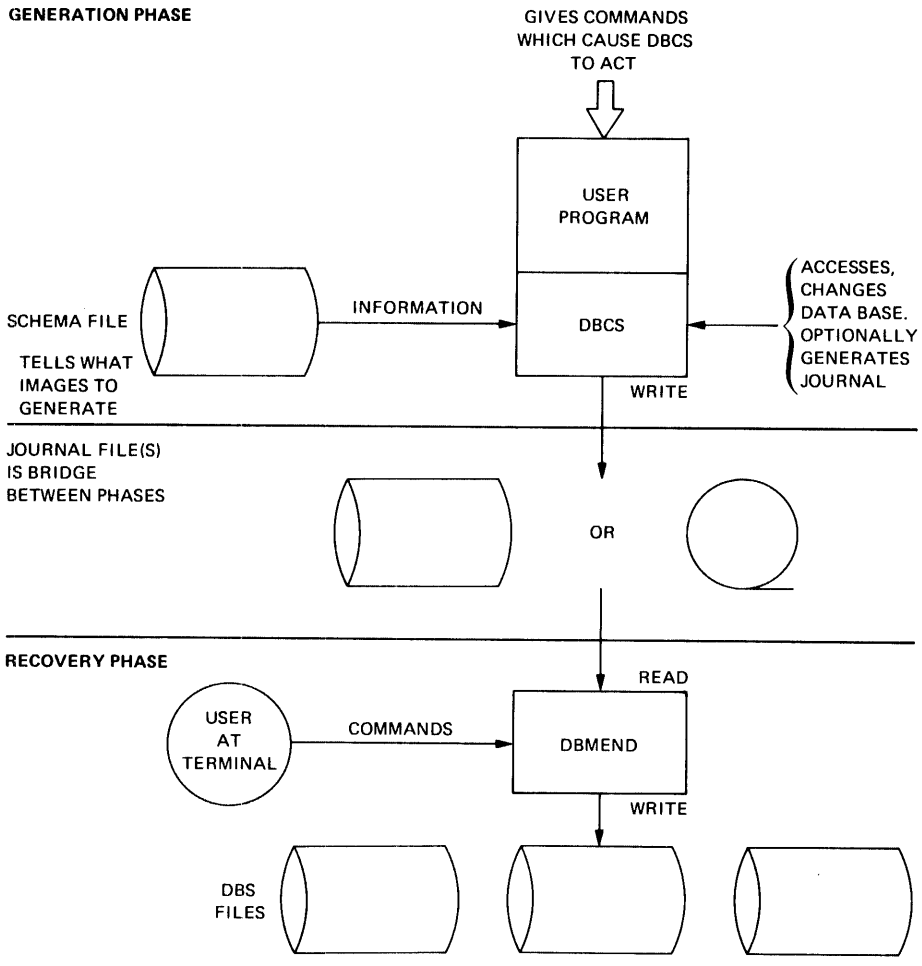


Figure 6-1 Generation and Usage of a Journal File

6.2.1 DBMS Journal

A journal is a file on disk or magnetic tape that contains images of pages from the data base plus additional information that delimits these pages. It is created during the execution of application run-units that open permanent areas in the update usage-modes. Each piece of information in a journal contains a run-unit ID, which enables you to isolate the changes made by a particular run-unit when you are using DBMEND. A journal file can be specified in one of two ways. You can include the BACKUP clause in the desired area DMCL entries or the application program can request BEFORE and/or AFTER images in a subprogram call. (See Section 2.3 in the *DBMS Programmer's Procedures Manual.*)

The additional information in the journal file will be some or all of the following:

1. A label block, which is the first block on each reel (a disk journal is one reel). The label block contains the schema-name, the run number, the reel number and the date and time when the run was started.
2. Command headers and trailers when images are ordered by command or constitute default transactions. (See Section 2.2.7.1.) They serve to delimit the BEFORE/AFTER data base images. They contain the DML command type that changed the data base page(s) between them and an index. The index is the number of the DML command in the journal file. DBCS increments it automatically each time the application program executes an updating DML command.
3. Transaction headers and trailers, which are optionally included by the program. These headers and trailers contain a transaction name and a transaction index. The transaction name identifies the type of transaction that occurred. The index uniquely identifies the occurrence of the transaction.
4. Comments that are optionally included by the program. These comments can contain any ASCII text and can occur anywhere in the file. This text is printable.
5. Data that is optionally included by the program. This data can be in any data mode (e.g., data-aggregate, SIXBIT, etc.) because it is transferred to the journal as is. This is non-printing text.

If concurrent run-units have opened areas in UPDATE usage-mode, and are using the journal file, the file is shared among them. (See also Section 2.5, which discusses simultaneous update.)

Because of the extra work involved in maintaining a shared journal, run-units should inform DBCS whether or not they intend to share the journal file. Before opening an area, therefore, the run-unit should execute the following statement:

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

If the journal is to be shared, the run-unit omits the keyword EXCLUSIVE. (Refer to Section 3 of the *Programmer's Procedures Manual* for the full description of the OPEN JOURNAL statement.)

Should a run-unit omit the OPEN JOURNAL statement, DBCS will simulate such a statement as follows:

- If the run-unit is opening an area in EXCLUSIVE or PROTECTED UPDATE usage-modes, an OPEN JOURNAL USAGE-MODE UPDATE is simulated. The journal is not shared.

- If the run-unit is opening an area in UPDATE usage-mode, an OPEN JOURNAL USAGE-MODE UPDATE is simulated. The journal is shared.

6.2.1.1 Appending/Overwriting the Journal – When a run-unit opens a journal file, DBCS must decide whether (1) it should append to the journal, (2) start at the beginning and overwrite pre-existing information, or (3) abort because the journal is in an undefined state.

DBCS overwrites the journal only if it has been informed that the existing information on the journal file is no longer needed. A journal file is no longer needed if

- the images it contains have been merged into the data base (see the DBMEND UNLOAD and COMPLETE commands), or

- you have indicated that all run-units have successfully completed execution.

By using the CLOSE JOURNAL statement, the last or only active run-unit can cause DBCS to alter the journal's label page to indicate that the journal is no longer needed. (See Section 3 of the *Programmer's Procedures Manual* for a full description of the CLOSE JOURNAL statement.)

DBCS appends to the journal when the journal's label page indicates it is appropriate to do so. Appending will occur under the following circumstances:

- yours is the only run-unit, and the previous run-unit had successfully executed a CLOSE RUN-UNIT statement.
- a concurrent run-unit has dequeued the data base, which it had maintained exclusively for the duration of either a command or a transaction. (See Section 2.5 on simultaneous update.)
- DBCS aborts the journal with exceptions xx61 or xx63 if one or more run-units have previously aborted and their images have not been merged into the data base. Should this occur, delete the journal file or use DBMEND to correct the situation.

6.2.2 DBMEND Functions

The three main functions of DBMEND allow you to:

1. merge BEFORE and/or AFTER images into the data base,
2. obtain an abstract of the journal file, and
3. adjust the Area Status Record of the DBS file.

Each of these functions is described below. The DBMEND commands relevant to the functions are mentioned with them, but are described in detail in Section 6.2.3.

6.2.2.1 Merging BEFORE/AFTER Images into the Data Base – To perform page recovery for the data base, you merge images from the journal file into the data base. You use BEFORE images to restore the data base to an earlier state. This would be necessary if the application program had put erroneous data in the data base, or if the system had crashed and the state of the data base were unknown. You use AFTER images to take the changes from the journal file and merge them into the data base (i.e., bring it to a later state). AFTER images are only necessary if you have to bring an old copy of the data base up to date.

You can merge all BEFORE or AFTER images from the journal file, or merge just images from selected portions of the journal file. You define the limits of the merge operations with the START and END commands. Refer to Section 6.2.4 for more information about start and end boundaries.

You can also specify that areas of the data base be excluded from the merge operation.

To merge BEFORE or AFTER images from a complete journal file, you should perform the following steps. You should have some knowledge of the contents of the journal file. You can gain this by performing an abstracting operation (see Section 6.2.2.2).

1. Run DBMEND (DBMEND).
2. Open the journal file (JOURNAL).
3. Rewind the journal file, if necessary (REWIND).
4. Open areas of the data base (OPEN).
5. Specify the portion of the journal that you will use by setting the start and end boundaries (START and END).
6. Specify that BEFORE or AFTER images will be merged (MERGE BEFORE or MERGE AFTER).
7. Close the open data base areas (CLOSE).
8. Close the journal file, if desired (UNLOAD).
9. Exit from DBMEND (^C) or start another operation.

An example of a merge of BEFORE images is as follows:

```
.R DBMEND
/JOURNAL MTA1:SCHEX
/REWIND
/OPEN MARKETING, PERSONNEL
/START
/END
/MERGE BEFORE
/CLOSE MARKETING, PERSONNEL
/UNLOAD
/^C
```

The above example assumes that the journal file is complete (i.e., no crash occurred and the application program ran to completion), but that the changes made by the program were incorrect. Thus, you wanted to merge all the BEFORE images in the journal file (i.e., undo the run entirely). However, many times you would only want to merge some of the images from the journal file. One reason could be that the system crashed during execution of the application program leaving the data base in an undefined state. You might then wish to restore only those data base pages at the end of the journal file.

If the system crashed before an application program was able to run to completion, the journal file, if it is on magnetic tape, is still sitting at the point where the crash occurred and does not have an end-of-file marker. In addition, some of the data in the temporary journal file may still be unmerged. Use the COMPLETE command to merge the as yet unmerged data and to put an end-of-file marker on the tape. You must not rewind the tape before you put the end-of-file on it because you will lose the current position and cause DBMEND to take longer to find the end of the usable data. You can rewind the tape after you have used the COMPLETE command, but you probably should not if you are going to work with the end of the journal file. Note that if you do not rewind the tape, you must specify the name of the schema to DBMEND by means of the SCHEMA command because the schema name is at the beginning of the tape in the label block and DBMEND will not have seen it.

The steps for merging BEFORE images back to the last completed DML command in the journal file and for putting the end-of-file on the tape are as follows:

1. Run DBMEND (DBMEND).
2. Open the journal file (JOURNAL).
3. Put the end-of-file on the journal, if necessary (COMPLETE).
4. Specify the start and end boundaries (START and END).
5. Specify the name of the schema, if necessary (SCHEMA).
6. Open the data base areas using the FORCEOPEN command to adjust the Area Status Record. Refer to Section 6.2.2.3 (FORCEOPEN).
7. Specify that BEFORE images will be merged (MERGE BEFORE).
8. Close the open data base areas (CLOSE).
9. Close the journal file, if desired (UNLOAD).
10. Exit from DBMEND (^C) or start another operation.

An example of merging in the above case is as follows:

```
.R DBMEND
/JOURNAL MTA1:SCHEX
/COMPLETE
/SCHEMA SCHEM03
/START LAST
/END
/FORCEOPEN MARKETING, PERSONNEL
/MERGE BEFORE
```

```
/CLOSE MARKETING, PERSONNEL
/UNLOAD
/^C
```

If you wish to merge AFTER images up to the last completed DML command or transaction, you can perform the following steps:

1. Run DBMEND (DBMEND).
2. Open the journal file (JOURNAL).
3. Put the end-of-file on the journal, if necessary (COMPLETE).
4. Rewind the journal (REWIND).
5. Specify the start and end boundaries (START and END).
6. Open the data base areas using the FORCEOPEN command to adjust the Area Status Record. Refer to Section 6.2.2.3 (FORCEOPEN).
7. Specify that AFTER images will be merged (MERGE AFTER).
8. Close the open data base areas (CLOSE).
9. Close the journal file, if desired (UNLOAD).
10. Exit from DBMEND (^C) or start another operation.

An example of this procedure is as follows:

```
.R DBMEND
/JOURNAL
/COMPLETE
/REWIND
/START
/END LAST
/FORCEOPEN MARKETING, PERSONNEL
/MERGE AFTER
/CLOSE MARKETING, PERSONNEL
/UNLOAD
/^C
```

When performing merge operations, you may want to keep track of the operation so that if anything happens (e.g., a system crash) you can continue from where the merge left off rather than duplicate some of the merge. You can use the TRACE command to cause DBMEND to type the command or transaction ID that has just been processed. An example of using the TRACE command and continuing after a crash is as follows:

```
.R DBMEND
/JOURNAL MTA1:SCHEX
/START
/END
/TRACE
/MERGE BEFORE
[BACK TO COMMAND 134]
[BACK TO COMMAND 133]
      .
      .
      .
[BACK TO COMMAND 12]
(system crash)
.R DBMEND
/JOURNAL MTA1:SCHEX
/START
/END 12
/TRACE
/MERGE BEFORE
      .
      .
      .
```

6.2.2.2 Obtaining Abstracts – Before performing merge operations, you may need to know what is in the journal file. The ABSTRACT command is used for this purpose.

Like the MERGE command, the ABSTRACT command requires that you specify start and end boundaries. You can also control the type of information in the abstract by means of the DISPLAY command. You can get such information as data-base-pages updated, transaction names and indexes, DML command indexes, any comments added to the file and the ID of the run-unit that performed these actions. Refer to the DISPLAY command for a complete description of this information. If you do not give a DISPLAY command, DBMEND will put command indexes, transaction names and indexes, data base page numbers, and comments into the abstract.

Because an ABSTRACT command need not deal with data base pages, you need open data base areas only when page information is to be displayed.

Often, you would want an abstract of the entire journal file. You can obtain one by performing the following steps.

1. Run DBMEND (DBMEND).
2. Open the journal file (JOURNAL).
3. Specify start and end boundaries (START and END).
4. Specify the information in the abstract, if desired (DISPLAY).
5. Specify that an abstract will be performed (ABSTRACT).
6. Close the journal file, if desired (UNLOAD).
7. Exit from DBMEND (^C) or start another operation.

An example of this procedure is as follows:

```
.R DBMEND
/JOURNAL MTA1:SCHEX
/START
/END
/DISPLAY ALL
/ABSTRACT SCHX.ABS
/UNLOAD
/^C
```

Note that an abstract should not be performed on a journal file that does not have an EOF. That is, if the program creating the journal file terminated abnormally so that there is no EOF on the end of the magnetic tape, you should put the EOF on the tape (using the COMPLETE command) before performing the abstract.

Once you use the COMPLETE command to put the EOF on the tape, however, you can get an abstract of the journal. The procedure to do this is as follows. (Note that if you do lose the location of the end of the usable data, running an abstract until a data error occurs is a good way of finding that location again.)

1. Run DBMEND (DBMEND).
2. Open the journal file (JOURNAL).
3. Put an EOF on the tape (COMPLETE).
4. Rewind the journal (REWIND).
5. Specify start and end boundaries (START and END).
6. Specify the information in the abstract, if desired (DISPLAY).
7. Specify that an abstract will be performed (ABSTRACT).
8. Close the journal file, if desired (UNLOAD).
9. Exit from DBMEND (^C) or start another operation.

An example of this procedure follows.

```
.R DBMEND
/JOURNAL MTA1:SCHEX.JRN
/COMPLETE
/REWIND
/START
/END
/ABSTRACT SCHX
/UNLOAD
/^C
```

6.2.2.3 Adjusting the Area Status Record – If an application program terminates abnormally, the Area Status Record of each data base file that was opened for update by that program could contain erroneous data. If this occurs, whether or not a journal file exists, you will want to fix the Area Status Record so that the DBS file can be accessed again by application programs or by DBMEND. You would use the FORCEOPEN command in this case so that DBMEND will override the bad information in the Area Status Record.

To use DBMEND to perform just this function, you can follow the steps below. Note that you do not have to open a journal file; but if you do not, you must specify the name of the schema to DBMEND.

1. Run DBMEND (DBMEND).
2. Specify the name of the schema (SCHEMA).
3. Open the areas in the data base (FORCEOPEN).
4. Close the areas in the data base (CLOSE).
5. Exit from DBMEND (^C) or start another operation.

An example of this procedure follows.

```
.R DBMEND
/SCHEMA SCHE03
/FORCEOPEN ALL:EXCLKEY
/CLOSE ALL
/^C
```

Note that many of the earlier examples (Sections 6.2.2.1 and 6.2.2.2) inherently accomplished Area Status Record cleanup because it is the sequence of FORCEOPEN and CLOSE commands that is relevant even if there are other commands intervening.

6.2.3 DBMEND Commands

The commands to DBMEND are described on the following pages in alphabetical order, each command starting a new page. Command line format is described before the commands because it applies to all of them. DBMEND uses the SCAN program to process the command lines; the following rules conform to SCAN requirements.

As long as no other keyword within the same context has the same abbreviation, a keyword can be abbreviated. The context of a keyword is the set of all other keywords that can appear in the same place. Thus, all commands are in one context, and the set of possible arguments to each command is each in a separate context. For example, you must spell out ALL when you give it as an argument to the CLOSE command because you could use an area-name in the same context. On the other hand, you can use MERGE A instead of MERGE AFTER, even though A is the abbreviation for the ABSTRACT command, because A is in the context of an argument to the MERGE command.

Spaces and tabs are ignored in a command line; and you must separate items in lists (e.g., a list of area-names) with commas. You can use either upper or lower case in a command line. However, DBMEND will treat a value as all upper case in a comparison. For example, if you give an area-name as lower case, DBMEND will use its upper-case equivalent when checking the name against the area-name in the schema. All commands and their modifiers are shown in upper case in this chapter.

DBMEND will only accept names composed of alphanumeric characters. Thus, if you use a special character such as hyphen (-) in a name, you must enclose the entire name in double quotation marks so that DBMEND will treat it as a name. For example, CLOSE A-B will not be acceptable, but CLOSE "A-B" will.

You can continue a command on another line by placing a hyphen as the last (rightmost) character of the line before the carriage return. The next line is then considered to be a continuation of the hyphen-ended line.

You can exit from DBMEND by entering a CTRL-C (^C). If you wish to return to DBMEND after exiting, you can issue the CONTINUE system command (or the REENTER system command to abort unwanted output) as long as you have not run any program that destroyed the core-image.

DBMEND will accept indirect command files. That is, you can store one or more sets of commands in a file and then submit the file to DBMEND by putting its file specification preceded by an at sign (@) on the command line.

```
.R DBMEND
/@file specification
```

The file specification is the device, filename, extension and project programmer number of the indirect file.

The DBMEND functions are merging, abstracting, and DBS file Area Status Record manipulation. For ease of reference, Table 6-2 gives a list of the commands with the DBMEND functions to which they are relevant. The minimum abbreviation for each command is shown in parentheses following the command.

Table 6-2
DBMEND Commands

Command		DBMEND Function
ABSTRACT	(A)	abstracting
BUILD	(B)	abstracting
CLOSE	(C)	all
COMPLETE	(CO)	abstracting, merging
DISPLAY	(D)	abstracting
END	(E)	abstracting, merging
EXCLUDE	(EX)	merging
FORCEOPEN	(F)	all
JOURNAL	(J)	abstracting, merging
LABEL	(L)	abstracting, merging
MERGE	(M)	merging
NOTRACE	(N)	merging
OPEN	(O)	merging, abstracting
POSITION	(P)	merging, abstracting
REELS	(R)	merging, abstracting
REWIND	(REW)	merging, abstracting
SCHEMA	(SC)	all
START	(S)	abstracting, merging
TRACE	(T)	merging
UNLOAD	(UN)	abstracting, merging

ABSTRACT

Use the ABSTRACT command to cause DBMEND to output an abstract of information from the journal file according to your specified boundaries. The ABSTRACT command has the format:

ABSTRACT file specification

The file specification is the device, filename, and extension of the file for the abstract. If you omit the device, DBMEND assumes DSK. You must specify the filename (if applicable), but you can omit the extension. If so, DBMEND assumes .ABS.

Before you issue the ABSTRACT command, you must set the boundaries for the abstract by specifying them in the START and END commands. If you have not previously given a DISPLAY command during the current execution of DBMEND, the abstract will include DML commands, page numbers, transactions, and text blocks. Otherwise, the types of information specified in the DISPLAY command are included in the abstract. Refer to the START and END commands for more information about boundaries and to the DISPLAY command for more information about what you can place in an abstract.

Example

```
/ABSTRACT DSK: JRNL1.ABS
```

BUILD

Use the BUILD command to cause DBMEND to generate an image-mode file from the journal file according to your specified boundaries. An image-mode file is one in which the data from the journal is put into the file exactly as it appears in the journal. You would use it to transfer non-textual data from the journal to a separate file. The BUILD command has the format:

BUILD file specification

The file specification is the device, filename, and extension of the file. If you omit the device, DBMEND assumes DSK. You must specify the filename (if applicable) but you can omit the extension. If you do, DBMEND assumes .ABS.

Before you issue the BUILD command, you must set the boundaries by specifying them in the START and END commands. If you have not previously given a DISPLAY command during the current execution of DBMEND, the file will include DML commands, page numbers, transactions, and text blocks. Otherwise, the types of information specified in the DISPLAY command are included in the file. Refer to the START and END commands for more information about boundaries and to the DISPLAY command for more information about what you can place in an image-mode file.

Example

```
/BUILD DSK:CKPNT.ABS
```

CLOSE

Use the CLOSE command to cause DBMEND to close those areas in the data base that are in an opened or excluded state. The CLOSE command has the following formats:

1. CLOSE area-1[,area-2] . . .
2. CLOSE ALL

The first format causes DBMEND to close the specified area or areas. The second format causes DBMEND to close all open and excluded areas.

The CLOSE command does not affect the status of the journal file, only the status of areas in the data base. You should issue it for all open (or excluded) areas before you terminate execution of the DBMEND program. The closing of an open area causes its Area Status Record to be set to 0.

Example

```
/CLOSE MARKETING, PERSONNEL
```

COMPLETE

Use the COMPLETE command to cause DBMEND to mark the end of the current journal. For magnetic tape, this means DBMEND places two end-of-file (EOF) markers at the appropriate position in the journal file. For disk, this means that DBMEND sets the appropriate journal page headers to denote EOF. The formats of the COMPLETE command are

Format 1

```
COMPLETE
```

Format 2

```
COMPLETE n
```

Use Format 1 of the COMPLETE command when an application run-unit that was appending to a magnetic-tape journal terminates abnormally, and the journal file is not closed. This form of the command causes uncopied data from the temporary journal file to be merged and end-of-file markers to be placed at the appropriate place.

Use Format 2 to truncate the journal file immediately after the page you specify. This form of the command causes an end-of-file marker to be placed immediately after the page (n) you specify.

When using the COMPLETE command, issue it immediately after you identify the journal with a JOURNAL command. After execution of the COMPLETE command, the journal file is still positioned at its end; it is open and available for all operations. DBMEND returns to command level.

Example

```
.R DBMEND
/JOURNAL MTA1:SCHES.JRN
/COMPLETE
```

DISPLAY

Use the DISPLAY command to define the kind of information that you want to be included in an abstract of the journal file. The formats for the DISPLAY command are:

1. DISPLAY [COMMAND | TRANSACTION | PAGENUM | TEXT | DATA] [{ BEFORE | AFTER } | HEADERS | RUNUNITID]

2. DISPLAY ALL

The arguments to the DISPLAY command have the following meanings.

1. COMMAND – The abstract will identify each DML command executed and its index.
2. TRANSACTION – The abstract will contain start-of-transaction and end-of-transaction lines for each user-defined transaction in the journal file.
3. PAGENUM – The abstract will contain the page number and area of the pages that are in the journal file. If you specify neither BEFORE nor AFTER, DBMEND will assume both types of images.
4. TEXT – The abstract will contain any text comments placed in the journal file.
5. DATA – The abstract will contain any non-textual data that the application program put in the journal file. If you specify DATA for an ABSTRACT command rather than for a BUILD command, DBMEND will include the following message in the abstract.

   [NON-TEXTUAL INFORMATION]

   If you specify DATA for a BUILD command, DBMEND will copy the non-textual data as is to the file specified in the BUILD command.
6. BEFORE – The abstract will contain page-related information for BEFORE images only.
7. AFTER – The abstract will contain page-related information for AFTER images only.
8. HEADERS – The file specified in a BUILD command will contain the logical block header as well as the information normally given with the other arguments. You can only specify this argument if you are going to use the BUILD command.
9. RUNUNITID – The abstract will show which run-unit generated transactions, text, and commands.
10. ALL – The abstract will contain all of the information that the arguments in column one specify.

If you do not give an argument, DBMEND will assume COMMAND, TRANSACTION, PAGENUM, and TEXT. A DISPLAY command with no argument is internally generated by DBMEND when it is started. It remains in effect until you issue a DISPLAY command.

Example

```
/DISPLAY COMMAND, PAGENUM
```

Use the END command to define the right boundary (i.e., the boundary closer to the end of the file) of the portion of the journal file from which DBMEND will take an abstract or on which DBMEND will perform a merge. The formats of the END command are:

1. END
2. END { integer-1 / LAST } : [run-unit ID]
3. END { literal-1 / * } : { integer-2 / LAST } : [run-unit ID]

Integer-1 is a DML command index. Literal-1 is a transaction-name as it is found in the user-specified transaction header. Integer-2 is a transaction index, also from the transaction header. Run-unit ID identifies the run-unit that generated the commands and transactions.

The first format specifies the end of the journal. The second specifies that the rightmost boundary will be either the specified command index or LAST, which means the last completed DML command in the file. The third format specifies that the rightmost boundary will be one of the following:

1. The specified transaction with the specified index (i.e., name:index).
2. The last completed transaction of the specified type (i.e., name:LAST).
3. The first encountered transaction with the specified index (i.e., *:index).
4. The last transaction in the file (i.e., *:LAST).

The boundary you specify in an END command must be to the right of (i.e., closer to the end of the journal than) the boundary you specify in the START command at the time that you give a MERGE or ABSTRACT command. You must explicitly give at least one pair of start and end boundaries for each journal that you manipulate during any execution of DBMEND. If you do not include run-unit IDs in the boundary indicators DBMEND ignores the run-unit IDs in the journal file when attempting a boundary match.

Refer to Section 6.2.4 for an overview of start and end boundaries, tape direction, and tape positioning.

EXCLUDE

Use the EXCLUDE command to specify that some or all areas of the data base will not be affected by subsequent merges that would otherwise affect them. The formats for the EXCLUDE command are:

1. EXCLUDE area-1[,area-2] . . . [:key-1]
2. EXCLUDE ALL [:key-1]

The first format allows you to exclude one or more areas from journal file operations. The second format allows you to exclude all areas from journal file operations. For either format, you must specify the privacy key for EXCLUSIVE UPDATE (key-1) if you have specified a privacy lock in the schema. If you specify several areas together in one EXCLUDE command, the areas must all have the same privacy lock (or none). If the areas that you wish to use have different privacy locks, you must use separate EXCLUDE commands to exclude them.

The EXCLUDE command has two purposes.

1. To pass over certain areas if they cannot be restored or if they do not need to be restored.
2. To cause DBMEND to try to perform a merge operation without the data base being disturbed. If the merge operation is successful, you can then issue the CLOSE ALL command, open some or all areas, and perform the actual merge.

Example

```
/EXCLUDE PERSONNEL:EXCLK
```

FORCEOPEN

Use the FORCEOPEN command to open one or more areas of the data base for EXCLUSIVE UPDATE after a crash has occurred. This is necessary because the Area Status Record after a crash or abnormal termination could have a value other than 0. A DML OPEN command could not open the area in such a case. The FORCEOPEN command causes DBMEND to override the bad Area Status Record. The formats of the FORCEOPEN command are:

1. FORCEOPEN area-1[,area-2] . . . [:key-1]
2. FORCEOPEN ALL [:key-1]

You use the first format to open one or more areas and, if necessary, override a bad Area Status Record. You use the second format to open all areas and, if necessary, override a bad Area Status Record in each. For either format, key-1 specifies the privacy key for EXCLUSIVE UPDATE. You must provide it if you specified a privacy lock in the schema. If you specify several areas together in a FORCEOPEN command, the areas must all have the same privacy lock (or none). If the areas that you wish to use have different privacy locks, you must use separate FORCEOPEN statements to open them.

```
/FORCEOPEN SALESAREA:EXULK
```

JOURNAL

Use the JOURNAL command to cause DBMEND to open the specified journal file. Its format is:

JOURNAL file specification

The file specification contains the device and filename of the journal file in the form:

dev:file[p,pn]

If you omit the device, DBMEND assumes DSK. You cannot omit the filename for a disk file. You should not use a filename extension because DBMEND always assumes .JRN. If you omit the project programmer number, DBMEND uses that of the logged-in user.

You must give the JOURNAL command before any other command that performs actions on the journal file. You can open only one journal file at any one time for DBMEND operations.

If you are opening a magnetic-tape journal and are using Format 1 of the COMPLETE command, specify the filename of the temporary journal file as well as the device field. This enables DBMEND to locate your temporary journal file and perform the COMPLETE command.

Example

/JOURNAL JRN:SCHEX

LABEL

Use the LABEL command to cause DBMEND to type the label of the reel of the journal file currently being used. Its format is:

```
LABEL
```

You can give the LABEL command at any time during DBMEND processing as long as a journal file is open. The information in the label includes:

1. schema-name
2. reel number
3. run number
4. date/time journal was created

A journal file on disk always consists of one reel.

Example

```
/LABEL
SCHEX
REEL 3
RUN 2
30-DEC-76 1115:25
```

MERGE

Use the MERGE command to cause DBMEND to copy either BEFORE or AFTER images from the journal file into the data base. DBMEND merges only those images between the start and end boundaries. The MERGE command has the format:

MERGE {BEFORE / AFTER}

Before you give the MERGE command, you must specify start and end boundaries.

MERGE BEFORE causes DBMEND to replace pages in the data base with BEFORE images taken from the journal file. The BEFORE images are copies of the pages before they were changed by the application program. When performing a merge of BEFORE images, DBMEND reads the journal file backward from the end (right) boundary to the start (left) boundary. This is because the earliest BEFORE images are at the beginning of the file. If DBMEND were to read the file forward, it might incorrectly merge later images of the same pages into the data base, replacing the correct earlier images.

MERGE AFTER causes DBMEND to replace pages in the data base with AFTER images from the journal file. The AFTER images are copies of the pages after they were changed by the application program. When performing a merge of AFTER images, DBMEND reads the journal file forward from the start boundary to the end boundary. This is because the latest AFTER images are at the end of the file.

Examples

```
/MERGE BEFORE
/MERGE AFTER
```

NOTRACE

Use the NOTRACE command to cause DBMEND not to trace the progress of subsequent merge operations. Its format is:

```
NOTRACE
```

You need only specify the NOTRACE command if you had previously given the TRACE command and you wish to stop tracing. DBMEND, when you start it, assumes that no tracing will be done.

Example

```
/NOTRACE
```

OPEN

Use the OPEN command to open one or more areas in the data base. Its formats are:

1. OPEN area-1[,area-2] . . . [:key-1]
2. OPEN ALL [:key-1]

You can specify a maximum of 12 areas.

The first format causes the specified area or areas to be opened in EXCLUSIVE UPDATE mode. The second format causes all areas to be opened in EXCLUSIVE UPDATE mode. For either option, key-1 is the privacy key for EXCLUSIVE UPDATE. You must specify it if the schema declaration contains a privacy lock for the area or areas. If you specify several areas together in one OPEN command, the areas must all have the same privacy lock (or none). If the areas you wish to use have different privacy locks, you must use separate OPEN commands to open them. If you cannot open any one area, you may have to issue a CLOSE ALL to avoid deadly embrace (waiting for an area while the user of that area is waiting for one of the areas that you have open).

The OPEN command opens data base areas only. It does not affect the journal file.

Example

```
/OPEN "MARKETING-AREA", "PERSONNEL-AREA":EXKEY
```

POSITION

Use the POSITION command to cause DBMEND to move to the specified position in the journal file. You can also use it to request typeout of the current position in the journal file. Refer to Section 6.2.4.3 for more information about positioning. The formats of the POSITION command are:

1. POSITION $\left\{\begin{matrix}+n\\-n\\n\end{matrix}\right\}$
2. POSITION

The first format causes DBMEND to move to the page in the journal file specified by +n, –n or n. The arguments have the following meanings.

1. +n specifies the journal page relative to the beginning of the current reel of the journal file. Thus, POSITION +5 means the fifth journal page from the beginning of the current reel.
2. -n specifies the journal page relative to the end of the current reel of the journal file. Thus, POSITION -5 means the fifth journal page from the end of the current reel. Similarly, POSITION -1 means the last journal page of the reel.
3. n specifies the actual journal page number in the file. Thus, POSITION 5 means journal page 5.

The POSITION command will not cause reels to be changed. If DBMEND cannot find the specified position on the current reel, DBMEND makes journal motion directionless and issues an error message.

If DBMEND executes the POSITION command successfully, it sets journal motion to be for either direction. That is, DBMEND does not have to initialize direction whether the next requested motion is forward or backward.

The second format causes DBMEND to type the number of the current journal page in the file; if EOJ was reached, however, the position displays as 0 by convention. In most cases, the current journal page is not the page in which DBMEND is logically positioned, but the page immediately following it. This is because the current journal page is normally in the look-ahead buffer. However, if you specify a POSITION command with an argument and immediately follow it by a POSITION command without an argument, DBMEND will be logically positioned in the journal page in which it is physically positioned. For example, the sequence of commands:

```
POSITION -1
POSITION
```

will always cause DBMEND to type the number of the last journal page on the current reel.

Example

```
/POSITION -2
```

REELS

Use the REELS command to specify to DBMEND the number of reels in a multi-reel journal. Its format is:

REELS [n]

You must specify the argument to the REELS command as an unsigned integer constant. If you do not give an argument, DBMEND assumes 1. If you do not give the REELS command, DBMEND assumes that there is only one reel of the journal file.

Example

/REELS 3

REWIND

Use the REWIND command to position DBMEND at the beginning of the current reel of the journal file and to initialize forward motion. Its format is:

```
REWIND
```

If the journal file is on magnetic tape, the REWIND command causes the tape to be physically rewound and DBMEND's pointer to be set at the beginning of the tape.

If the journal file is on disk, the REWIND command causes DBMEND's pointer to be set at the beginning of the file.

Example

```
/REWIND
```

SCHEMA

Use the SCHEMA command to specify the name of the schema to which the journal file applies. Its format is:

SCHEMA schema-name

Use the SCHEMA command to specify the name of the schema to DBMEND when it is not known. DBMEND will not know the schema name if DBMEND has not yet read the journal page containing the label of the journal. For example, after abnormal termination of a run-unit that created a magnetic tape journal, you might not want to rewind the tape to find the label because you want to start merging from the end of the file (i.e., do a MERGE BEFORE).

You can also use the SCHEMA command to specify the name of the schema when you only want to use the FORCEOPEN command to adjust the Area Status Record in one or more DBS files. You would need the SCHEMA command in this case because you would not have opened a journal. You should give the SCHEMA command before the FORCEOPEN command.

Example

/SCHEMA SCHEX

START

Use the START command to define the left boundary (i.e., the boundary closer to the beginning of the file) of the portion of the journal file on which DBMEND will take an abstract or perform a merge.

The formats of the START command are:

1. START
2. START { integer-1 / LAST } : [run-unit ID]
3. START { literal-1 / * } : { integer-2 / LAST } : [run-unit ID]

Integer-1 is the DML command index. Literal-1 is a transaction-name as it is found in the user-specified transaction header. Integer-2 is a transaction-index, also from the transaction header. Run-unit ID identifies the run-unit that generated the commands and/or transactions.

Format 1 can mean one of two things. It will mean, to the ABSTRACT command only, the current position if journal direction is currently forward. In all other cases, it will mean the start of the journal file.

Format 2 specifies that the beginning boundary will be either the specified command index or LAST, which means the last completed DML command in the file.

Format 3 specifies that the start boundary will be one of the following:

1. The specified transaction-name with the specified index (i.e., name:index).
2. The last complete transaction of the specified name (i.e., name:LAST).
3. The first encountered transaction with the specified index (i.e., *:index).
4. The last transaction in the file (i.e., *:LAST).

The boundary that you specify in the START command must be to the left of (i.e., closer to the beginning of the file than) the boundary you specify in the END command at the time that you give a MERGE or ABSTRACT command. You must explicitly give at least one pair of start and end boundaries for each journal that you manipulate during any execution of DBMEND. If a run-unit ID is missing from a boundary indicator, DBMEND ignores the run-unit ID in the journal file when attempting a boundary match.

Refer to Section 6.2.4 for a description of start and end boundaries, tape direction, and tape positioning in the journal file.

Example

```
/START ADDCOM:4
```

TRACE

Use the TRACE command to cause DBMEND, during subsequent merge operations, to type a progress report after it processes the data base pages in each transaction (or DML command). The format of the TRACE command is:

TRACE

The message printed depends on whether the merge operation involves reading forward (MERGE AFTER) or backward (MERGE BEFORE). For forward motion the message is:

[THRU name]

For backward motion, the message is:

[BACK TO name]

In either case, name is either a transaction or a command name, depending on the type of boundaries that you set in the START and END commands.

Messages from the trace operation allow you to resume a merge after a crash with a minimum of remerging. This is because you can specify the appropriate boundary on the basis of the last trace message you received before the crash.

DBMEND does not perform tracing unless you explicitly give the TRACE command. Tracing remains in effect until you give the NOTRACE command.

Example

/TRACE

UNLOAD

Use the UNLOAD command to cause DBMEND to logically close the current journal and physically unload the current reel of a magnetic tape journal. Its format is:

```
UNLOAD
```

The UNLOAD command causes the journal file to become unavailable. This means that DBMEND (in addition to physically unloading a magnetic tape, if necessary) reinitializes all journal-related modes. These include tape motion, which DBMEND sets to be directionless; start and end boundaries, which it sets to null; and the current journal label, which it sets to null.

Specifying the UNLOAD command for a disk journal indicates the information in the journal is no longer necessary and can be overwritten. (The journal is placed in a done-with-state.)

Example

```
/UNLOAD
```

6.2.4 Boundaries, Direction, and Positioning of the Journal File

You must understand certain concepts if you want to perform relatively sophisticated operations with DBMEND. These include boundaries, direction of processing, and positioning within the journal file.

6.2.4.1 Start and End Boundaries – The most important concept about boundaries is that START defines the left boundary and END defines the right boundary (see Figure 6-2) in the sense that the boundary specified in the START command always defines the boundary closer to the beginning of the journal file, even if the direction of processing is backward. Similarly, the boundary specified in the END command is always the boundary closer to the end of the journal file.

If you wish to get an abstract or perform a merge on the entire journal, the start and end boundaries are relatively easy to determine. For an abstract or a merge of AFTER images, you need only specify START without an argument and END LAST. START without an argument means the beginning of the journal, and END LAST means the last completed DML command in the journal. If you wish just to merge BEFORE images from the end of the journal to the last completed DML command, you can specify START LAST and END without an argument. END without an argument means the end of the journal.

However, when you wish to merge or abstract arbitrary portions of the journal, you must have a more precise understanding of the arguments to the START and END commands.

When the direction of processing in the journal file is forward, the boundary indicators are always inclusive. That is, DBMEND begins with the DML command or transaction specified as the left boundary and finishes after the DML command or transaction specified as the right boundary. This is shown in Figure 6-2.

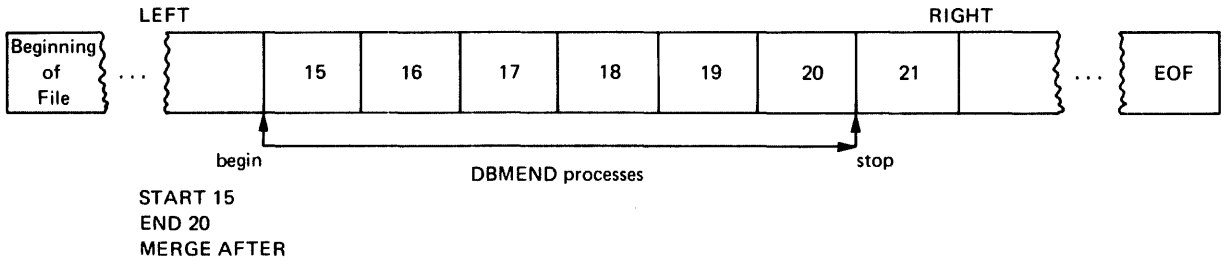


Figure 6-2 DBMEND Boundaries on Forward Processing

When direction of processing in the journal file is backward, however, the left boundary indicator is exclusive. That is, DBMEND finishes with the DML command or transaction immediately after the one specified as the left boundary and begins at the right of the DML command or transaction given as the right boundary. Boundaries on backward processing are shown in Figure 6-3.

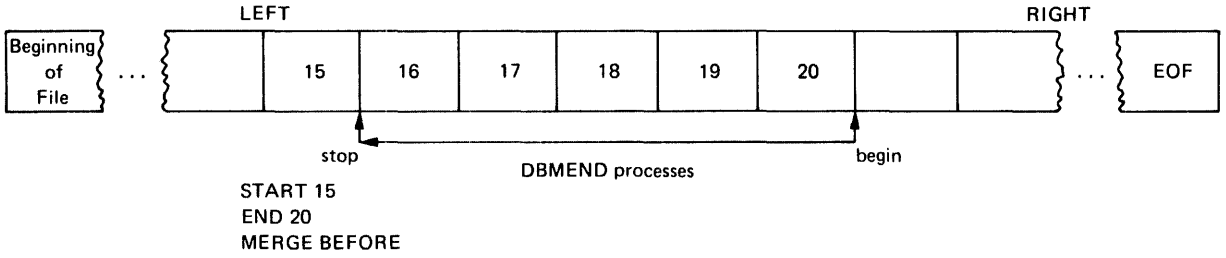


Figure 6-3 DBMEND Boundaries on Backward Processing

6.2.4.2 Direction of Motion – When processing the journal file, DBMEND must determine where it should be in the journal and in what direction the journal should be moving. Consequently, certain commands cause direction to be set to forward, others cause direction to be set to backward, and others cause the journal to become directionless. In addition, DBMEND may have to initialize journal direction as well as set it before beginning command processing. This means for forward motion that DBMEND is positioned at the beginning of the current reel, and for backward motion that DBMEND is positioned at the end of the current reel.

The ABSTRACT, BUILD, and MERGE AFTER commands initialize forward motion if the current direction is not forward. If the current direction is forward, the ABSTRACT, BUILD, and MERGE AFTER commands cause DBMEND to continue from where it is currently positioned. The REWIND command always causes forward initialization.

The MERGE BEFORE command initializes backward motion if the current direction is not backward. If it is backward, MERGE BEFORE causes DBMEND to continue from where it is currently positioned.

The COMPLETE and JOURNAL commands cause the journal to become directionless. That is, after one of these commands, DBMEND has no current direction and will always initialize direction with the next direction-setting command.

The POSITION command is special in that it sets direction to either forward or backward, depending on the next direction-setting command. In other words, after a POSITION command, MERGE AFTER or MERGE BEFORE will cause DBMEND to continue rather than initialize forward or backward motion.

A general exception to the rules for initialization of direction is that if the boundary to be processed first (i.e., the start boundary for forward motion and the end boundary for backward motion) is relative to the end of the current reel, the journal is always temporarily initialized at the end of the reel. For forward motion, this means a START command with any form of LAST (START LAST, START *:LAST, START n:LAST). For backward motion, this means an END command without an argument or an END command with any form of LAST (END LAST, END*:LAST, END n:LAST).

Table 6-3 lists those commands that affect direction along with the direction that each sets. Also shown is whether or not the command can initialize motion.

Table 6-3
DBMEND Commands Affecting Direction

Command	Direction	Initialization
ABSTRACT	forward	if necessary
BUILD	forward	if necessary
COMPLETE	directionless	no
JOURNAL	directionless	no
MERGE AFTER	forward	if necessary
MERGE BEFORE	backward	if necessary
POSITION	forward or backward	no
REWIND	forward	always

Note: Initialization is necessary if:

1. motion is currently directionless,
2. there is a change of direction, or
3. the end boundary for a merge of BEFORE images is relative to the end of the reel.

In summary, initialization does not occur when the current motion is the same as that required, or if START LAST is the boundary for an operation in the forward direction.

6.2.4.3 Positioning – DBMEND is inherently positioned in both a physical and logical sense. Physical positioning deals with I/O – the last journal page that has been read in. Logical positioning deals with logical blocks (e.g., command headers, text blocks); it is the location of DBMEND's conceptual pointer in the journal. Refer to Section 6.2.5 for more information about journal pages and logical blocks.

You specify start and end boundaries in terms of a logical position, but you specify arguments to the POSITION command in terms of a physical position. Thus, if you give POSITION 8, you cause DBMEND to physically position after the eighth journal page in the file, not the eighth command or transaction. Note that physically positioning to a page means that it becomes the last journal page on which DBMEND performed I/O. Also, as a side-effect of physically repositioning the journal, you reposition the journal logically. In particular, if the direction of motion becomes forward, the logical position is the first complete block at the beginning of the physical page specified in the POSITION command. Conversely, if direction of motion becomes backward, the logical position is the last complete block at the end of the physical page specified in the POSITION command.

The POSITION command is useful for positioning DBMEND in relation to the beginning or end of the current reel of the journal file. You can also use it to position DBMEND near to the journal page identified by a DBMEND error message. Lastly, when you are requesting an abstract of a temporary journal file, specify POSITION 2 because page 2 of the temporary journal file is really at an arbitrary place in the actual journal file.

6.2.5 Physical Aspects of the Journal File

6.2.5.1 Format of the Journal File – Physically, the journal file is divided into fixed-length pages. Each page contains 512 words, with the first six words reserved for a page header that describes the information in the page. Into each page, portions of one or more logical blocks are packed. Logical blocks are variable in length because they can be DML commands, transactions, BEFORE or AFTER images, comments, or data. Thus, a logical block can start in one journal page and continue into the next journal page. The format of a journal page is shown in Figure 6-4.

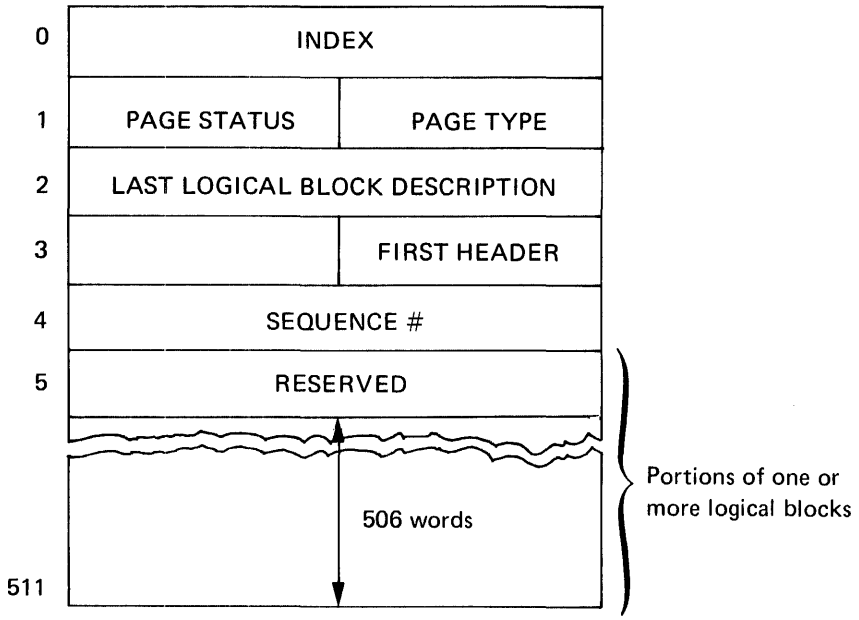


Figure 6-4 Format of a Journal Page

INDEX is the number of each page in the file. It is sequential, starting with 1.

PAGE STATUS describes the status of the page:

n<0 – end of the page is potential waste in the sense that the last n words on the page start a logical block that is completed on a later page.

$n \geq 0$ – n words at the end of the page are unused.

PAGE TYPE describes the type of information in the page:

0 – page is empty

1 – page contains data only

2 – page starts with a label

LAST LOGICAL BLOCK DESCRIPTION contains the size of the page's last logical block and its ID. A logical block ID can be one of the following:

1. the logical block is a BEFORE image
2. the logical block is an AFTER image
3. the logical block is a command header
4. the logical block is a command trailer
5. the logical block is a label
6. the logical block is a transaction header
7. the logical block is a transaction trailer
8. the logical block is information
9. the currency indicator has changed (DBCS use only)

FIRST HEADER contains the offset of the first complete logical block in the page, or is 0 if there is none.

SEQUENCE # contains a number that is incremented each time DBCS goes to the beginning of the file.

Each logical block header has the format shown in Figure 6-5.

CURR. SIZE	CURR. ID
PRV. SIZE	PRV. ID
RUNUNITID	

Figure 6-5 Logical Block Header

CURR. SIZE is the size of the current logical block.

CURR. ID is the ID of the current logical block.

PRV. SIZE is the size of the previous logical block.

PRV. ID is the ID of the previous logical block.

RUNUNITID is the ID of the run-unit that created the block.

The format of a logical information block is shown in Figure 6-6.

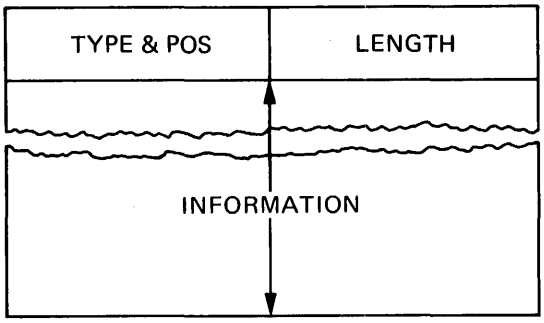


Figure 6-6 Format of an Information Block

TYPE & POS contain the type of information and whether the block is the first, middle, or last block of information.

Bit 0 describes the type of information:

0 – textual (printable) information

1 – non-textual (non-printable) information

Bits 1 through 3 indicate the position of the block:

Bit 1 is 1 if this is the first block of the information.

Bit 2 is 1 if this is a middle block of the information.

Bit 3 is 1 if this is the last block of the information.

If all three bits are 0, this block contains all of the information specified in one call to the JRDATA or JRTEXT subprogram in the run-unit.

Bits 4 through 17 are reserved for future expansion.

LENGTH is the length of this information block (in words for data or ASCII characters for text). Each block can have a maximum of 16 words.

If the logical block is a label block, which is the first block in the first page on a reel, the block has the format shown in Figure 6-7.

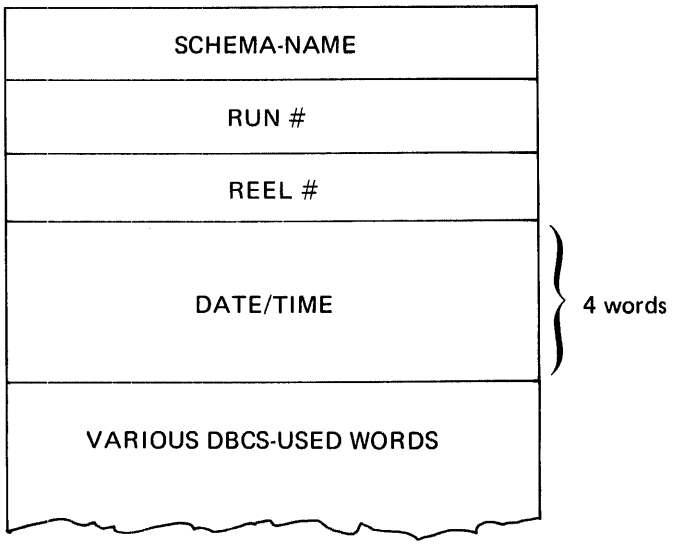


Figure 6-7 Format of a Label Block

SCHEMA-NAME is the name of the schema accessed by the application program creating the journal file.

RUN # is the number of the run that accessed the schema when creating the journal file.

REEL # is the number of the current reel.

DATE/TIME is the date and time when the journal file was created.

6.2.5.2 .TMP Files – DBMEND creates a temporary disk file for a magnetic tape journal file when it is processed in the backward direction. As the journal is read backward, chunks of it are placed in the temporary file and processed there. (The size of a chunk is 32 journal pages.) The name of the temporary file is nnnMND.TMP, where nnn is the job number. The temporary file resides in your directory. When the backward operation is complete, the .TMP file is deleted.

6.2.6 DBMEND Messages

The following are the messages put out by DBMEND. Those errors that begin with a question mark (?) are fatal; those with a percent sign (%) are warnings; and those enclosed in brackets are informational. Some data error messages are followed by an informational message in which you are asked to perform an action. This is described in Section 6.2.7.

[BEGINNING-OF-REEL n]
DBMEND has reached the beginning of the current reel while processing backward. It is followed by a message requesting action. See Section 6.2.7.

[CURRENT JOURNAL PAGE IS n]
This message is the reply to a POSITION command without an argument.

[JOURNAL END-OF-FILE AFTER JOURNAL PAGE n]
DBMEND found the end-of-file while processing forward in the journal. This message is followed by a message requesting action. See Section 6.2.7.

%MNDAAD AREA STATUS ALREADY DESIGNATED FOR area-name
You tried to open or exclude an area already opened or excluded. Close the area and try again.

?MNDAAE UNABLE TO: action area-name
This message covers a wide number of I/O actions that DBMEND cannot perform on an area.

?MNDAAO AREA area-name NOT IN "READY" STATE -- SEE "FORCE" IN MANUAL
The Area Status Record of the DBS file is bad and you must force an open by using the FORCEOPEN command.

?MNDAED ABNORMAL END OF DATA IN JOURNAL AFTER PROCESSING PAGE n
DBMEND reached an abnormal end of data because there is an error in the file. This message will be printed after you have typed A (attempt) after a data error and the attempt failed. See Section 6.2.7.

%MNDAPA ACTUALLY POSITIONED AT JOURNAL PAGE n: NON-CONSECUTIVE PAGES PASSED OVER
This message will be printed following a POSITION command if DBMEND finds that it is positioned at a journal page other than the one it expected.

%MNDBAD ILLEGAL REQUEST -- TRY AGAIN
You typed an illegal command or argument to DBMEND during error recovery. See Section 6.2.7.

?MNDBSF. BAD SCHEMA FILE -- REFERENCE IS symbol-name
When DBMEND tried to reference the schema file, it was unusable. Rebuild the schema file. If this message occurs again after the schema file is rebuilt, you should submit an SPR.

?MNDCLO IMAGE TO MERGE IN CLOSED AREA area-name
DBMEND attempted a merge operation, but the area was closed. Open the area and try again.

?MNDCOS. CANNOT OPEN/LOOKUP SCHEMA FILE schema-name
DBMEND cannot open the referenced schema file. Check that the correct name was used and that the schema is in an accessible directory.

?MNDCPJ CURRENT POSITION IN JOURNAL PAST INITIAL BOUNDARY

You set a beginning boundary that was beyond the current position in the journal. Either change the boundary or change the position in the journal.

?MNDDCA DATA CHANNELS ALL IN USE

DBMEND cannot open all files specified because no JFNs are available. Exclude some areas and try again.

?MNDDER JOURNAL DEVICE ERROR AFTER JOURNAL PAGE n

An error occurred on the device. This message is followed by a request for you to perform an action. See Section 6.2.7.

?MNDDTE JOURNAL PARITY ERROR AFTER JOURNAL PAGE n

A parity error occurred on the journal device. This message is followed by a request for action. See Section 6.2.7.

%MNDEMA EOF BOUNDARY FOR "MERGE AFTER" DANGEROUS

Specifying END without an argument as the boundary for MERGE AFTER could be dangerous if the end of the journal is incomplete.

%MNDEOD ENCOUNTERED EOD AFTER PROCESSING JOURNAL PAGE n BEFORE FINDING END BOUNDARY

DBMEND came to the end of the data in the journal file before it reached the end boundary.

%MNDFAE UNABLE TO: action

DBMEND was unable to perform the specified I/O action.

?MNDIAN INVALID AREA NAME area-name

You specified an invalid area-name. Try again with a valid area-name.

?MNDIBD INVALID BLOCK DESCRIPTOR ON JOURNAL PAGE n

The header of a logical block is bad. DBMEND types a request for action after this message. See Section 6.2.7.

?MNDIFB IMPROPERLY FORMED BOUNDARY VALUE

You entered a boundary value in the wrong form. The value must be LAST or an unsigned integer.

?MNDIFM INITIALIZE FORWARD TAPE MOTION FIRST -- EG. REWIND

You attempted to perform a LABEL command, but forward motion had never been initialized. REWIND to initialize forward motion.

?MNDIJD IMPROPER JOURNAL DEVICE -- NOT MTA OR DSK

You specified an improper device in the JOURNAL command. Use a correct device in the JOURNAL command.

?MNDIPI INVALID JOURNAL PAGE ID ON PAGE ADJACENT TO n

The header of a journal page is bad. DBMEND types a request for action after this message. See Section 6.2.7.

?MNDIPN INVALID DATA BASE PAGE n -- ENCOUNTERED ON JOURNAL PAGE n

DBMEND found a reference to an invalid data base page in the journal file.

?MNDIPX INVALID JOURNAL PAGE INDEX ON PAGE ADJACENT TO n

The header of a journal page is bad. DBMEND types a request for action after this message. See Section 6.2.7.

%MNDJCI JUST COMPLETED REEL'S LABEL INCONSISTENT WITH PREVIOUS REEL'S LABEL
While going backward, DBMEND found that the label of the reel it just finished does not agree with the label of the previous reel. This could mean that the merge operation put incorrect data in the data base.

?MNDJCN JOURNAL CONTAINS NO DATA
You attempted to work with an empty journal file.

?MNDJNI JOURNAL NOT YET IDENTIFIED
You attempted to perform an operation on the journal before giving a JOURNAL command. Issue a JOURNAL command and try again.

?MNDJPI JOURNAL POSITIONED INCORRECTLY AFTER PROCESSING LAST-RELATIVE BOUNDARY
This is a system error. Submit an SPR.

?MNDMOP MONITOR OR PROGRAM ERROR AFTER JOURNAL PAGE n IF ERROR RECURS
An error occurred in processing the journal file. If the error recurs, submit an SPR.

?MNDNAD NO ACCESSIBLE DATA IN LAST JOURNAL CHUNK -- SEE MANUAL
The last chunk of the journal that was read into the .TMP file (see Section 6.2.5.2) starts with bad data.

?MNDNLI NEW REEL'S LABEL INCONSISTENT WITH CURRENT REEL'S LABEL
When processing a multi-reel journal forward, DBMEND found that the next reel was not the one that should be next. DBMEND does not process the new reel, but asks for your action. See Section 6.2.7. Usually just mount the correct reel and try again.

?MNDNNZ NON-NUMERIC OR ZERO POSITION SPECIFIED
The argument to the POSITION command was incorrect. Try again with the correct argument.

?MNDNSB. NO SCHEMA BLOCK IN .SCH FILE -- REBUILD IT
DBMEND cannot find the schema block for the schema file. Rebuild the schema file to rebuild the schema block. If the problem recurs after the schema block is rebuilt, submit an SPR.

?MNDNSD NO SCHEMA NAME HAS BEEN DETERMINED/SUPPLIED AS YET
You attempted to access the data base before giving DBMEND the name of the schema either in the SCHEMA command or in the label of the journal. Supply the schema name and try again.

?MNDOVN START OR END VALUE UNINITIALIZED
You did not supply a value for the START or END command. Give the command with a value and try again.

?MNDPIF JOURNAL'S LABEL PAGE HAS IMPROPER FORMAT
This is a system error if the file identified in the JOURNAL command is a valid journal. If such is the case, submit an SPR.

?MNDPKI PRIVACY KEY INCORRECT FOR area-name
You supplied an incorrect privacy key for an area. Determine the correct privacy key and try again.

%MNDRNC RESUMED AFTER "SKIP" WITH NON-CONSECUTIVE PAGE n
When DBMEND skips a bad page, it still tries to keep track of the page number it should see next. This message can be ignored.

?MNDSEM START/END BOUNDARIES NOT OF SAME CATEGORY
The value in the start boundary is not of the same type as that in the end boundary. Correct one of the boundaries and try again.

?MNDSSC SPURIOUS SCHEMA COMMAND -- DIFFERENT NAME IN JOURNAL LABEL
DBMEND found that the schema name used in the SCHEMA command does not agree with the schema named in the label.

?MNDUPS UNREACHABLE JOURNAL POSITION SPECIFIED
You gave a value in the POSITION command that was not on the current reel.

?MNDURP UNABLE TO RESTORE DATA BASE PAGE IN AREA area-name
DBMEND cannot put a BEFORE or AFTER image into the specified area. If this error recurs, submit an SPR.

?MNDWCP WILDCARDING IS PROHIBITED
DBMEND does not accept wildcard formats in its commands.

?MNDWFT UNABLE TO READ JOURNAL PAGE n WHILE FILLING .TMP FILE
When writing chunks to the .TMP file (see Section 6.2.5.2) DBMEND could not read a journal page. DBMEND asks for action after this message. See Section 6.2.7.

[NO CURRENT JOURNAL PAGE]
You entered a POSITION command without an argument, but DBMEND has no current journal page.

[TYPE CONTINUE TO RESUME EXECUTION]
After a new reel of the journal is mounted, you must type CONTINUE to the system to return to DBMEND processing.

6.2.7 Incremental Error Recovery

Certain data errors can cause DBMEND to stop, type a message, and wait for a response because the error may be recoverable. The classes of error are:

1. end of file on a reel
2. unreadable data on a journal page
3. bad control information in a logical block.

Following the message describing the error, DBMEND tells you the replies that you can make. The acceptable replies are among the following:

A – attempt
S – skip
M – mount
Q – quit
E – end

A (attempt) can be a reply to DBMEND when it has encountered a journal page containing bad data in its look-ahead buffer. A tells DBMEND to continue processing the good data in the current buffer. However, if the DBMEND operation (i.e., a merge or abstract) is not completed by the time the good data is exhausted, DBMEND then gives another message:

?MNDAED ABNORMAL END OF DATA

which is not recoverable.

S (skip) can also be a reply when DBMEND finds a bad journal page. S tells DBMEND to skip over (the rest of) the bad journal page and continue processing beyond that page at the first good, complete logical block it sees. Figure 6-8 shows the data that DBMEND reads before and after you reply with S.

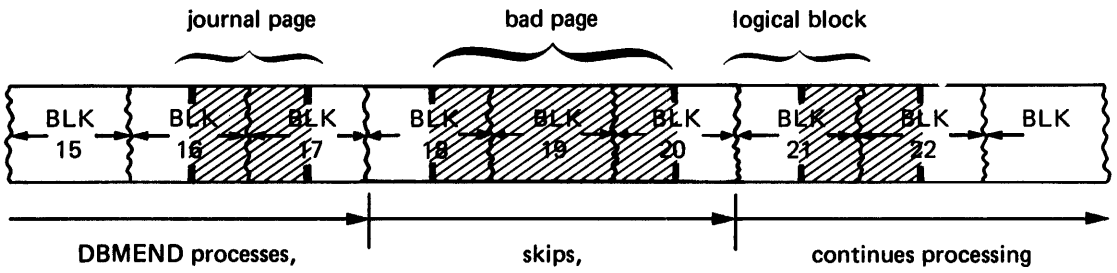


Figure 6-8 DBMEND Actions When Skipping Bad Data

Note that S (skip) should be used cautiously. When data is skipped, DBMEND has no way of knowing if some of the data skipped is crucial to the integrity of the data base – only you can know this. Thus, you should only use S during a merge operation if you know that the data that will be skipped is not crucial.

M (mount) is a reply that should be given when DBMEND reaches logical end-of-reel. Logical end-of-reel normally occurs when DBMEND detects EOF. (See example 2 below.) It also applies if there is a spurious bad journal page at the end of a reel that is really not part of the journal and should be ignored. M tells DBMEND that another reel of the journal is available so that DBMEND will unload the current reel and continue processing with the new reel when it is mounted.

Q (quit) is a reply that can be given whenever you do not want to try to recover from a data error. Q tells DBMEND to give up and return to command level.

E (end) is a reply that is applicable when a reel is being initialized at the end for a merge of BEFORE images. It should be given when a bad journal page is at the end of a reel and you know that the real end of data on the reel is the good page before that bad page. E tells DBMEND that it should ignore the bad page and treat the previous page as the last good page on the reel.

Examples

1. .R DBMEND

```
      .
      .
      .
/MERGE AFTER
[JOURNAL END-OF-FILE AFTER JOURNAL PAGE 168]
[TYPE Q(UIT) or (M)OUNT]
M
[TYPE CONTINUE TO RESUME EXECUTION]
.CONT
      .
      .
      .
```

2. .R DBMEND

```
      .
      .
      .
/MERGE BEFORE
?MNDDTE JOURNAL PARITY ERROR AFTER JOURNAL PAGE 42
[TYPE (Q)UIT, (A)TTEMPT, (S)KIP]
S
      .
      .
      .
```

6.3 THE DAEMDB PROGRAM

The DAEMDB program is a utility that copies data from the temporary journal file (on disk) to magnetic tape. In effect, it allows you to perform magnetic-tape journalling at your facility. The program can be run under OPSER, as one of the subjobs controlled by OPSER, or it can be run as a normal timesharing job from a terminal.

Once you have specified magnetic-tape journalling (via the JOURNAL statement of the DMCL) DBCS initializes your journal usage via an IPCF message to DAEMDB. DBCS also establishes a temporary journal file on disk by replacing the device field in your journal specification with DSK. When journalling begins, DAEMDB reads data from the temporary journal file on disk and writes the data onto the magnetic-tape unit chosen to hold the journal file.

When DAEMDB copies the information from the temporary file onto magnetic tape, it also retains the data base long enough to modify the journal label page thereby informing DBCS that this information in the temporary journal is no longer needed. The next time DBCS writes to the journal, it starts at the beginning – overwriting the information already copied to magtape. In this way, DAEMDB behaves like a concurrent run-unit operating within the simultaneous update facility.

There are a number of advantages to the way that DAEMDB operates (as a background task) and to the fact that the temporary journal file is an ordinary disk journal. These are as follows:

1. When DAEMDB reaches end-of-reel on a magtape, the application run-unit performing journalling can continue processing while the current reel is being replaced. The temporary journal file (on disk) simply continues to grow. When the reel has been replaced and DAEMDB indicates readiness to continue, the information on the temporary disk file is copied to magtape as usual.
2. If DAEMDB should abort, the application run-unit performing journalling can continue processing. The reason is that the run-unit is actually writing to the temporary journal on disk.
3. When you are using DBMEND to perform page recovery on the data base, you do not need to involve the magnetic-tape journal if the images you want are still on the temporary disk journal file. You can specify the temporary journal file specification to the DBMEND JOURNAL command. (See Section 6.2 for a discussion of DBMEND.)
4. Messages pertaining to management of the magnetic tape reels are not seen by the application run-units performing journalling. Only the job controlling DAEMDB is informed when a new reel is needed or an error has occurred.

6.3.1 Initiating Magnetic-Tape Journalling

To initiate magnetic-tape journalling for your data base, you must make the appropriate specification in the JOURNAL statement of the DMCL. You must do this to be able to run DAEMDB at your facility. (See Chapter 3 of this manual for a description of the DMCL JOURNAL statement.)

You can initiate magnetic-tape journalling by specifying that the device in the journal-file specification is MTAn. If you specify a unit number, that particular magnetic-tape unit will be suggested to the system operator. The following example shows this method.

```
JOURNAL IS MTA2: MYJOUR.
```

If you do not specify a unit number, the system operator freely chooses the magnetic-tape unit. The following example shows this method.

```
JOURNAL IS MTA: MYJOUR.
```

In either case, be sure to include the complete file specification so that the temporary journal file specification can be derived. Remember that DBCS creates the temporary file on disk by replacing the device field in your specification with DSK. Note that if DAEMDB is not running as a privileged job, the protection code of the temporary journal file should be such that DAEMDB will be able to access it.

If you, the DBA, have specified magnetic-tape journalling and the service is not available (exception condition 0946), application run-units are given the opportunity to select disk journalling. DBCS types the following message, including the appropriate DAEMDB interaction code.

```
%DBSODJ ONLY DISK JOURNAL FEASIBLE (DAEMDB CODE x)
```

(Note that the DAEMDB interaction codes are listed in Table 6-4.) You can then use the TOPS-10 command ASSIGN to associate the journal name with a disk device. Then type CONTINUE. The following example shows this procedure.

```
.
.
.
%DBSODJ ONLY DISK JOURNAL FEASIBLE (DAEMDB CODE 1)
[TYPE CONTINUE TO RESUME]
.ASSIGN DSK: MTA2:
.CONTINUE
.
.
.
```

If an application run-unit wishes to continue processing when DBCS discovers that magtape service is not available, that run-unit should include a USE procedure in its code for exception-condition 0946 (magtape service unavailable). The run-unit can also associate exception-condition 0967 (unable to initialize magtape service) to the same USE procedure. The USE procedure should be placed before the first OPEN statement in the application run-unit code. (See Chapter 3 of the *Programmer's Procedures Manual* for a description of USE procedures.) The USE procedure can call the subprogram JMDISK, which automatically specifies the journal device to be DSK.

Table 6-4 lists the DAEMDB interaction codes and describes the meaning of each. These codes will appear to the application run-units as part of the %DBSODJ messages.

Table 6-4
DAEMDB Interaction Codes

Code	Meaning
1	DAEMDB unavailable (SHUTDOWN already given)
2	Unable to open magnetic tape unit
3	Unable to open temporary journal file
4	Improper device type after DEFINE
5	DBCS software error during initialization
6	JMDISK already done once
7	Journal Quota already reached

6.3.2 Running DAEMDB

You can run DAEMDB as a normal timesharing job from a terminal or as one of the subjobs controlled by the OPSER program. The remainder of this section presents information applicable to both. Section 6.3.2.1 discusses the procedures for running DAEMDB as a timesharing job from a terminal. Section 6.3.2.2 discusses the procedures for running

DAEMDB as one of the subjobs under OPSER. To fully understand the procedures described in these two sections, also read Section 6.3.3, which describes the DAEMDB commands and their functions.

Remember also that DAEMDB may be managing a number of journal files. (It can manage up to eight at one time.) Each journal is associated with a specific schema-name. DAEMDB relies on its knowledge of the schema-name associated with each journal file to work properly. This reliance is expressed in terms of volume-ids. A volume-id has the form

aaaabb

where aaaa represents the first letters of the schema-name and bb represents the current magnetic-tape reel number. For example, if the name of your schema is MYSKE and you have a 2-reel magnetic-tape journal, the volume-id of the first reel would be MYS01; that of the second reel would be MYS02. You can, therefore, pre-label each reel; in this way, you ensure that the system operator mounts the correct (pre-labelled) reel.

On the other hand, if you have many schemas at your installation, be sure that the first letters of each schema-name are unique. Each schema-name associated with a journal file as represented by the volume-id will then be unique.

The volume-id thus informs the system operator (or other DAEMDB user) which magnetic tape reel to mount. It also has another use. When the ABORT, CREATE, and MOUNT commands are being issued, the volume-id (which is a part of these commands), also informs DAEMDB which journal to make current. (Remember DAEMDB may be managing many journals.) Unless DAEMDB has just issued an action-required message, therefore, containing a volume-id, you must specify the volume-id to DAEMDB with the ABORT, CREATE, and MOUNT commands.

DAEMDB issues an action-required message when a magnetic-tape is initially opened and each time the current reel requires replacing. The message looks this way.

[ACTION REQUIRED FOR VOLUME volume-id ON MTAx]

The volume-id in the message always refers to the current reel. If you want to mount a new volume, you should specify the new reel via a volume-id when you issue the MOUNT command.

In addition, when you are replacing a magnetic-tape reel, do not consider the new reel to be a part of your journal file until DAEMDB responds with the following message:

[VOLUME volume-id NOW INITIALIZED]

Examples of the circumstances discussed in this section, as they would occur during processing, are shown in the following sections.

6.3.2.1 Running DAEMDB as a Timesharing Job – To run DAEMDB from a terminal, type R DAEMDB in response to the DECsystem-10 prompt character (.); follow it with a carriage return (↵).

.R DAEMDB↵

You can then continue with the commands described in Section 6.3.3 just as you would with any program from your terminal. Your job will receive messages from DAEMDB.

6.3.2.2 Running DAEMDB Under OPSER – To run DAEMDB under OPSER, read Chapter 4 of the *DECsystem-10 Operator's Guide*, which explains OPSER conventions. (Note that DAEMDB would be one of the subjobs controlled by OPSER.) Especially note that DAEMDB requires a 2-letter code to function under OPSER.

The assigned code is DB. To run DAEMDB under OPSER, perform the following sequence:

.R OPSER	Run OPSER. Hit Carriage Return. OPSER responds with an asterisk (*). Note that OPSER responds with an exclamation point if another subjob is running.
*:SLOGIN 1/2	Login subjob DAEMDB under [1,2].
*:DEF DB=	Associate subjob n with the name DB.
*DB–R DAEMDB	Tell DB to run DAEMDB.

DAEMDB is now defined as a subjob under OPSER, logged-in, and started.

When DAEMDB is started, it behaves as if it had just performed a polling (the initial polling period is 100 seconds); it will then not poll the journals again until the number of seconds you specified as the frequency of polling has passed.

When a magnetic tape is initially opened, DAEMDB issues an action-required message. Typically, you respond with the MOUNT command. If the action-required message from DAEMDB has been preceded by a magnetic tape or temporary journal file error message, however, you might respond with the MOUNT, the ABORT or the RETRY command. The following example shows receipt of an error message during the process of copying data from the temporary journal file to magnetic tape.

```
DAEMDB in process
      .
      .
      .
      .
[DABJME END-OF-TAPE REACHED FOR MTA2 JOURNAL OF MYSKE]
[ACTION REQUIRED FOR VOLUME MYS01 on MTA2]
```

Respond by mounting a new tape reel. Then type

```
MOUNT MTA2:MYS02
      .
      .
```

Wait until DAEMDB responds informing you that the new reel is officially part of the journal. The DAEMDB response looks this way.

```
[VOLUME MYS02 NOW INITIALIZED]
      .
      .
Continue processing.
```

(Note that if the system crashed before printing the above message, you must mount MYS01 when issuing the COMPLETE command to DBMEND.)

6.3.3 DAEMDB Commands

Table 6-5 lists the DAEMDB commands and their acceptable abbreviations. The commands ABORT, EXIT, and STOP cannot be abbreviated. They must be typed out in full.

Table 6-5
DAEMDB Commands and Acceptable Abbreviations

Command	Abbreviations
ABORT	ABORT
CREATE	CR
CURRENT	CU
EXIT	EXIT
GO	G
HELP	H
MOUNT	M
POLL	P
RESET	RES
RETRY	RET
[NO] SHUTDOWN	NOS or SH
STOP	STOP
THRESHOLD	T
WHAT	W

The commands to DAEMDB are described in detail on the following pages. They are in alphabetical order; each command starts on a new page.

ABORT

Use the ABORT command to stop processing of the magnetic tape portion of the journal. The ABORT command has the following format:

```
ABORT [device] [volume id]
```

The ABORT command is used in response to a request from DAEMDB that its operator perform an action. DAEMDB makes this request because of a magnetic tape error or because of end-of-tape.

Example

```
DAEMDB> ABORT
```

CREATE

Use the CREATE command to pre-associate a magnetic tape unit to the schema for which journalling is being performed. The format for the CREATE command is

CREATE [device] [volume id]

The device is MTA. The volume id has the form aaaabb where aaaa represents the first letters of the schema name; bb represents the current reel number.

An application run-unit requesting this journal will not experience a real-time wait while the operator mounts a tape.

Example

```
DAEMDB> CREATE
```

CURRENT

Use the CURRENT command to type out the current setting for each mode. The CURRENT command has the following format.

```
CURRENT
```

The CURRENT command is intended to be used as a tuning command. This means that once DAEMDB has informed you what the current polling and threshold settings are, you can change them if you feel they need modifying.

Example

```
DAEMDB> CURRENT
```

EXIT

Use the EXIT command to return to monitor level. The format is

```
EXIT
```

The EXIT command clears all active queue requests before actually exiting. This means that application run-units will not be left in a blocked state merely because DAEMDB execution was terminated.

You cannot continue the DAEMDB program after typing EXIT.

Example

```
DAEMDB> EXIT
```

GO

Use the GO command to continue processing after you have typed the STOP command. The format of the GO command is

```
GO
```

Example

```
DAEMDB> GO
```

HELP

Use the HELP command to print a table of DAEMDB commands. The format for the HELP command is as follows.

HELP

Example

DAEMDB> HELP

MOUNT

Use the MOUNT command to mount a new magtape. The MOUNT command format is as follows.

MOUNT [device] [volume id]

Use the MOUNT command in response to a DAEMDB request for action on a volume.

Example

DAEMDB> MOUNT

POLL

Use the POLL command to define and initiate a new polling period. The format of the POLL command is as follows.

POLL number-of-seconds

Once each number-of-seconds you specify, DAEMDB polls the temporary journal files on disk for data not yet copied to the magnetic-tape journal file. The default polling period is 100 seconds.

Polling minimizes the resources DAEMDB uses; DAEMDB remains inactive except when it copies from the temporary journal file. The POLL command allows you to control the frequency with which DAEMDB polls the journals it is managing for new data to copy.

Note that a longer polling period means less DAEMDB activity and longer temporary journal files. A shorter period means more CPU and I/O accesses, but shorter temporary journal files.

Example

```
DAEMDB> POLL 50
```

RESET

Use the RESET command to simulate starting DAEMDB. The format of the RESET command is as follows.

```
RESET
```

This command is intended to give the operator a means of getting DAEMDB back to a defined state without closing the table of journals DAEMDB is managing.

Example

```
DAEMDB> RESET
```

RETRY

Use the RETRY command to cause DAEMDB to ignore the error it reported and to repeat the operation that failed. The format of the RETRY command is

RETRY [device] [volume-id]

The device argument is optional; provide it for clarity. The volume-id is required unless DAEMDB has just issued a prompt for this volume.

The RETRY command prevents DAEMDB from aborting upon occurrence of spurious I/O errors.

Example

```
DAEMDB> RETRY
```

SHUTDOWN

Use the SHUTDOWN command to tell DAEMDB (1) not to accept any new journals, and (2) to exit when all the active magnetic-tape journals have been closed by the application run-units. The formats for the SHUTDOWN command are as follows.

Format 1

```
SHUTDOWN
```

Format 2

```
NOSHUTDOWN
```

If you specify format 2, you will take DAEMDB out of a shutdown state allowing it to accept new journals.

Example

```
DAEMDB> SHUTDOWN
```

STOP

Use the STOP command to return to DAEMDB command level. The format for the STOP command is as follows.

```
STOP
```

Using the STOP command puts DAEMDB in a TTY-wait state. This allows you time, for example, to fix a magnetic tape drive that may not be working properly.

Example

```
DAEMDB> STOP
```

THRESHOLD

Use the THRESHOLD command to specify when (in terms number-of-pages) DAEMDB should copy information from the temporary journal files on disk to magnetic tape. The format for this command is

THRESHOLD number-of-pages

The initial threshold is 50 pages. When the temporary journal file on disk has reached the number of pages you indicate, DAEMDB copies the information onto the magnetic-tape journal and modifies the journal's label page to inform DBCS that this information is no longer needed. The next time DBCS writes to the temporary journal, it starts at the beginning, overwriting this information.

The purpose of a high threshold (large number-of-pages) therefore would be to minimize the number of times DAEMDB retains the data base exclusively to modify the temporary journal's label page.

It follows then, that a small threshold would, over the run of an application program, increase the number of ENQUEUEs and DEQUEUEs that program will require. This, in turn, will tend to increase CPU usage for the single user (that application program opening an area in EXCLUSIVE UPDATE usage-mode).

Example

```
DAEMDB> THRESHOLD 30
```

WHAT

Use the WHAT command to type the state of each journal DAEMDB is managing. The format of this command is as follows:

```
WHAT
```

Example

```
DAEMDB> WHAT

SCHEMA          MTA         USE-CNT          COMMENT

SIMUL           MTA1          2

ORDENT          MTA2          1              LOCKED

DAEMDB>
```

Note that this example also shows the DAEMDB response to the WHAT command.

6.3.4 Performing Page Recovery with a Magnetic-Tape Journal

When the journal file is on magnetic tape and you are using DBMEND to perform page recovery, be sure to observe the following rules.

1. Specify the same file specification to the DBMEND JOURNAL command as was specified by the aborted application run-unit(s). This allows DBMEND to automatically locate the journal temporary file (on disk) as well.
2. Specify the DBMEND REELS command if you have a multi-reel journal file.
3. Specify Format 1 of the DBMEND COMPLETE command to cause DBMEND to merge any unmerged data on the journal temporary file with the magnetic tape journal. This will be necessary if the system crashes during an active run-unit and all the data in the temporary journal file has not been copied to the magnetic tape journal.

   Note that when the magnetic tape is in an unknown position (and possibly positioned after the end-of-data), you should also specify the DBMEND REWIND command before specifying the COMPLETE command.
4. Proceed as you would with a disk journal. Refer to Section 6.2 for a complete description of DBMEND.

The procedure for using DBMEND with a magnetic tape journal is as follows:

1. Run DBMEND (DBMEND).
2. Open the journal file (JOURNAL).
3. Specify the number of reels (REELS). Only for multi-reel journals.
4. Merge any unmerged data on the journal temporary file with the magnetic-tape journal and put an EOF on the journal (COMPLETE).

   .
   .
   .

   [Error Message indicating cannot complete]
5. Rewind the journal. (REWIND). Because the journal was in an unknown position and possibly positioned after the end-of-data.
6. Merge any unmerged data on the journal temporary file with the magnetic-tape journal and put an EOF on the journal (COMPLETE).

The following example shows use of DBMEND with a magnetic-tape, multi-reel journal left in an unknown position such that completion will not occur without rewinding.

```
.R DBMEND
/JOURNAL MTA1:SCHEX
/REELS3
/REWIND
/COMPLETE
    .
    .
[ERROR MESSAGE]
    .
    .
/REWIND
/COMPLETE
    .
    .
```

6.3.5 DAEMDB Messages

This section lists the various messages you can receive from the DAEMDB program; briefly explains possible causes for each; and, where appropriate, suggests responses for each. Those errors that begin with a question mark (?) are fatal (in some way, they terminate processing); those that begin with a percent sign (%) are warnings; and those enclosed in brackets are informational.

?DABAMB UNACCEPTABLE ABBREVIATION OF COMMAND

You have used an unacceptable abbreviation of a DAEMDB command. See Table 6-5 for a list of commands and their acceptable abbreviations.

%DABBAD INAPPLICABLE REQUEST

You have attempted a response (either RETRY or MOUNT) to a prompt when no prompt was pending. You may also have attempted to mount a magtape when a RETRY or an ABORT would have been appropriate.

?DABCRE "CREATE" REQUIRES BOTH MTA AND VOLUME ID.

You have specified the create command without specifying the device and volume id. Both are required.

[VOLUME volume-id NOW INITIALIZED]

The magnetic tape reel you have mounted can now be considered to be a part of the journal. This means that DAEMDB has fully processed your MOUNT command.

%DABIMC INCONSISTENT MESSAGE CODE RECEIVED

This message indicates a spurious IPCF message.

?DABJAC JOURNAL ALREADY CREATED

The journal has been created.

[DABJME xxx FOR volume-id, JOURNAL OF schema-name]

This message informs you that end-of-reel (or some other magnetic-tape error) has been reached for the volume-id shown. The message can occur during process of copying from the temporary journal file on disk to the magtape journal. The message is followed by an action-required message. In response, you can issue the RETRY, MOUNT, or ABORT commands.

%DABJNO MESSAGE TO CLOSE NOT-OPEN JOURNAL SENT

A message has been sent to close the journal when the journal has been already closed. This indicates a possible software error or an attempt to perform an action that is prohibited.

?DABJOM COULD NOT OPEN magtape unit

The attempt to open the magnetic-tape unit has been unsuccessful. If this message was the response to your issuing a CREATE command, repeat the CREATE command. If it was the response to your issuing a MOUNT command, and repeating the MOUNT does not solve the problem, issue the ABORT command.

?DABJSE PAGE SEQUENCING ERROR FOR JOURNAL OF schema-name

This message can indicate a software error in the interaction between DAEMDB and DBCS. If the error recurs, send an SPR.

[DABJTE xxx FOR TEMP JOURNAL OF schema-name]

This message can occur during the process of copying data from the temporary journal file on disk to the magnetic-tape journal. The xxx indicates the particular disk error that has occurred. The message is followed by an action-required message. In response, you can issue the RETRY or the ABORT commands.

?DABJQA JOURNAL QUOTA ALREADY REACHED

Initialization of the magnetic tape journal has failed because DAEMDB can manage only eight journal files at one time. The application run-unit receives the DAEMDB interaction code number 7 (see Table 6-4) in the DBCS %DBSODJ message. The run-unit can associate the journal file with a disk device, or it can have made provision for such problems by having included a USE procedure in its code for exception-conditions 0946 and 0967 such that processing continues. See Section 6.3.1.

?DABJUN JUNK AT END OF COMMAND LINE

The last portion of your command line contains incomprehensible information, change it.

[ACTION REQUIRED FOR VOLUME volume id ON schema-name]

This message informs you that an action is required on the volume-id specified; the message follows the [DABJME, ?DABJSE, and [DABJTE messages. The latter messages can occur during the process of copying from the temporary disk journal to magnetic tape.

?DABNUM NUMBER WAS EXPECTED

The number specified with the POLL and THRESHOLD commands must be greater than 0.

?DABOJT UNABLE TO OPEN JOURNAL TEMP OF schema-name

Initialization of the temporary journal file on disk has failed. The application run-unit receives the appropriate DAEMDB interaction code in the DBCS %DBSODJ run-time message and can take necessary steps. See Section 6.3.1.

?DABPTL POLL-PERIOD GREATER THAN 600

You have specified a polling frequency that is more than 600 seconds. This is not acceptable. Change the frequency of polling.

?DABSER DAEMDB ERROR -- MESSAGE WILL REPEAT IF CANNOT FIX SELF

Repetition of this message indicates a problem with DAEMDB. Rerun DAEMDB.

[POLL= xxx, THRESHOLD= xxx]

You will receive this message in response to your CURRENT command request. The message tells you the frequency of polling and the number of pages after which DAEMDB copies the temporary journal to magtape.

%DABSQR SPURIOUS QUEUE REQUEST RECEIVED

This message indicates a problem in the interrupt system; however, processing continues.

?DABVOL UNKNOWN VOLUME ID SPECIFIED

You have specified an unidentifiable volume-id. Repeat the command and include a valid volume-id.

APPENDIX A
RESERVED WORDS

The following words are reserved in DBMS. Reserved means that you cannot use these words as user-created names. Refer to the *COBOL Programmer's Reference Manual* for COBOL reserved words. Those words in the list below preceded by 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
AREAS
**ARNAM
ASCENDING
ASSIGN
AT
AUTOMATIC

-B-

BACKUP
BEFORE
BIN
BINARY
BIND
BUFFER
BY

-C-

CALC
CALL
CHAIN
CLOSE
CLOSED
COMMAND
COMPILE
COMPLEX
COPY
COUNT
CURRENCY
CURRENT

-D-

DATABASE-KEY
DBKEY
DEC
DECIMAL
DELETE
DELETR
DESCENDING
DIRECT
DISPLAY
DISPLAY-6
DISPLAY-7
DISPLAY-9
DUPLICATES

-E-

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

-F-

FIND
FIND0
FIND1
FIND2
FIND3
FIND4
FIND5
FIRST
FIXED
FLOAT
FOR
FROM

-G-

GET
GETS

-I-

IF
IMAGES
IN
INSERT
INSRT
INTERCEPT
INTO
INVOKE
IS

-J-

JBTRAN
JETRAN
JMAFT
JMBEF
JMBOTH
JMNAME
JMNONE
JOURNAL
JRSYNC
JSTRAN

-K-

KEY

-L-

LAST
LINKED
LOCATION
LOCK

-M-

MANDATORY
MANUAL
MEMBER
MODE
MODIF
MODIFY
MOST
MOVE
MOVEC

-N-

NAME
NEXT
NOT
NOTE

-O-

OCCURRENCE
OCCURS
OF

ONLY
OPEN
OPEND
OPTIONAL
ORDER
OTHERS
OWNER

-P-

PAGE
PIC
PICTURE
PRIOR
PRIVACY
PROTECTED

-R-

RANGE
REAL
RECMEM
RECMO
**RECNAM
RECORD
RECORD-NAME
RECORDS
RECORDS-PER-PAGE
RECOWN
REMOV
REMOVE
RETRIEVAL
RPP
RUN-UNIT

-S-

SAVESS
SBIND
SCHEMA
SECTION
SELECTION
SELECTIVE
*SENTENCE
SET
SETCON
SETDB
SETS
SIZE
SORTED
STATS
STATUS
STORE
STORED
SUB-SCHEMA
SUPPRESS
SYSCOM
SYSTEM

-T-

TEMPORARY
TENANT
TEXT
THRU
TO
TYPE

-U-

**UNDEF
UNSET
UPDATE
UPDATES
USAGE
USAGE-MODE
USE
USING

-V-

VIA

-W-

WITHIN
WORDS

APPENDIX B
SCHEMA ERROR MESSAGES

?DDLADI ASC/DESC PHRASE INCOMPATIBLE WITH ORDER ALWAYS AND ORDER SORTED DBKEY

You specified an ASCENDING or DESCENDING phrase for a member in a set that has ORDER ALWAYS or ORDER SORTED BY DATABASE-KEY. Delete the ASCENDING/DESCENDING phrase or change the order.

%DDLANA ASSIGNED NAME NEVER APPEARS IN AREA STATEMENT

You specified an area-name in the DMCL but never used it in a schema area entry. Create an area entry for the area or let SCHEMA ignore the error.

%DDLASI. ALL MEANINGLESS SWITCHES ARE IGNORED

SCHEMA ignores all switches except /CREATE and /NOCREATE.

?DDLDPS DUPLICATE PSEUDONYM

You used the same pseudonym twice. Change one of them and try again.

?DDLDUP DUPLICATE name ENCOUNTERED

You used the same name twice. Change one of them and try again.

?DDLELW. ENCOUNTERED word WHILE action name

This is a general purpose message. SCHEMA has encountered one of the words in one of the phrases in the DMCL or DDL while performing some action (or expecting a name). Correct the phrase and try again.

?DDLFTL FILE SPEC COMPONENT TOO LONG

One of the components of one of the file specifications is too long. Correct the component and try again.

%DDLICI ILLEGAL CHARACTER IN INPUT ON LINE n

You have a character that is not allowed. Correct the character and try again or let SCHEMA ignore the error.

?DDLIFP ILLEGAL FACTORING IN A PICTURE

You used characters that are not allowed in the PICTURE phrase. Correct the phrase and try again.

?DDLILN IMPROPER LEVEL NUMBER

When specifying data-items you used an incorrect level number (i.e., not 02 in the schema record entry). Correct the level number and try again.

?DDLIMC DATA NAMES INCORRECTLY MATCH CALC KEYS FOR record-name

When you specified the data-items in the record, you did not include all of the CALC keys specified in the LOCATION MODE phrase. Correct the data-names and try again.

?DDLIOR INTEGER OUT OF RANGE n TO n INCLUSIVE

You specified a number that was out of the range previously specified (e.g., one of the numbers in the page range for a record is not in the page range for that area). Correct the number and try again.

?DDLIZL PICTURED ITEM HAS ZERO LENGTH

You specified a length of zero in the PICTURE phrase. Correct the length and try again.

?DDLKNM KEY NOT A DATA-NAME IN MEMBER RECORD TYPE

You specified a data-name as a sort key, but it was not part of the record. Add the data-name to the record or change the sort key.

?DDLKNO SOS KEY DOES NOT MATCH APPROPRIATE KEY OF OWNER RECORD TYPE

The key specified in the USING phrase in the SET OCCURRENCE SELECTION clause does not match the DIRECT or CALC key in the owner record. Correct the key and try again.

?DDLLDP LOGICAL LOCATION OF RECORD DEPENDS ON ITS PHYSICAL LOCATION -- BUT LOCATION MODE IS VIA-SET

You specified the location of a record in such a way that its logical location depends on its physical location. However, you also declared the record with LOCATION MODE VIA, which means that the logical location should not depend on the physical location. Correct either the dependence or the LOCATION MODE.

?DDLLND LOCATION MECHANISM DURING SOS MUST HAVE NO DUPLICATES ALLOWED

You specified a USING or ALIAS phrase in the SET OCCURRENCE SELECTION clause, but did not specify that duplicates were not allowed for the USING or ALIAS identifier. Correct the error and try again.

%DDLLTL LINE n TOO LONG

You specified a line that is too long. Shorten the line and try again or let SCHEMA ignore the error.

%DDLLSE LINE SEQUENCE NUMBER n NOT FOLLOWED BY TAB

The line sequence number that EDIT put on the line was not followed by a tab. SCHEMA assumes that the tab is missing and continues.

?DDLMSP MISSING SET PHRASE: MODE OR OWNER OR ORDER

You did not include one of the phrases MODE, OWNER, or ORDER in the set entry. Correct the set entry and try again.

?DDLNRP NO RECORDS-PER-PAGE SPECIFIED FOR AREA

You did not specify a RECORDS-PER-PAGE statement for an area. Add the statement and try again.

?DDLNTL DATA BASE NAME IS TOO LONG

One of the names used in the description of the data base is too long. Correct the name and try again.

?DDLOCD ONLY ASC/DESC WITHOUT DUPLICATES PHRASE COMPATIBLE WITH ORDER SORTED DUPLICATES

You cannot specify that duplicates are allowed in the ASCENDING/DESCENDING clause for a record in a set with ORDER SORTED DUPLICATES. Correct the ASCENDING/DESCENDING clause and try again.

?DDLOMD ASC/DESC PHRASE FOR ORDER SORTED [WITHIN] MUST HAVE "DUPLICATES" SUB-PHRASE

You must specify the DUPLICATES phrase in the ASCENDING/DESCENDING phrase for a record in a set that has ORDER SORTED WITHIN or ORDER SORTED. Correct the ASCENDING/DESCENDING phrase and try again.

?DDLOPF. OPEN FAILURE FOR filename

SCHEMA could not open one of the files (SCH or DBS). Check that the protection for the file or the directory is compatible or that the device is available. If the error persists, submit an SPR.

?DDLOSI OWNER-IS-SYSTEM INCOMPATIBLE WITH SOS

You cannot specify a SET OCCURRENCE SELECTION if the owner of the set is SYSTEM. Delete the SET OCCURRENCE SELECTION clause and try again.

?DDLPRO AREA'S PAGE RANGE OVERLAPS AN EARLIER PAGE RANGE

You specified a page range for one area that is within the range of another area. Correct the page range and try again.

?DDLPSI PAGE SIZE OF area-name INSUFFICIENT TO ACCOMMODATE record-name RECORDS

The records that you specified to be within an area will not fit because you do not have a large enough page size. Increase the page size or decrease the size of the record.

?DDLPSS PAGE SIZE OF area-name CANNOT ACCOMMODATE AREA-STATUS-RECORD AND SYSTEM RECORD

You specified such a small page size that the area-status-record on the SYSTEM record cannot fit into the area. Correct the page size and try again.

?DDLROD RECORD CAN CONTAIN ONE DISPLAY MODE ONLY

Only one DISPLAY (DISPLAY-6, DISPLAY-7, DISPLAY-9) mode can appear in all of the data-items in a record. Correct the data-items and try again.

?DDLSAF SCHEMA ACCESS FAILURE -- CHECK SCHEMA FILE BEFORE RETRYING

SCHEMA could not access the DDL file. Check the protection on the file and directory. Also check that the file is available. If the error recurs, submit an SPR.

?DDLSBR SET ENTRY FOR set-name MUST APPEAR BEFORE BEING REFERENCED IN SOS PHRASE

If you specify that SET OCCURRENCE SELECTION for a set is LOCATION MODE OF OWNER and the owner of the set has a LOCATION MODE of VIA, you must have previously described the set specified in the VIA phrase of the owner. Correct the schema file and try again.

%DDLSES ENCOUNTERED EOF -- SIMULATING END-SCHEMA STATEMENT

You did not end the schema with the END-SCHEMA statement or the file was truncated. SCHEMA assumes an END-SCHEMA statement.

?DDLSIE. SOURCE FILE INPUT ERROR -- TRY AGAIN

SCHEMA could not read the source file, probably because of a monitor error. Check the file and try again.

%DDLSIF SCALE FACTOR INAPPLICABLE TO FLOATING POINT ITEMS

You specified a scale factor for a floating pointer data-item. SCHEMA ignores the scale factor.

?DDLSTL STATEMENT TOO LONG OR "." MISSING.

You specified a statement that was longer than 120 characters or you left off the period. Correct the statement and try again.

?DDLTMO RECORD IS TENANT IN SET MORE THAN ONCE

You specified that a record is owner or member more than once in a set. Remove one description of the record from the set and try again.

?DDLTMR TOO MANY RECORD TYPES DEFINED IN SCHEMA

You specified more than 480 record types in the schema. Reduce the number of record types and try again.

?DDLTMS RECORD PARTICIPATES IN TOO MANY SETS

You have specified so many sets in which the record participates that the pointers will use more than the maximum of 512 words. Remove the record from some sets or delete some of the PRIOR and OWNER pointers in some sets.

?DDLTNS SPECIFIED DATA TYPE NOT SUPPORTED

You specified a data type that is not supported in DBMS. Change the data type and try again.

?DDLUCA UNABLE TO CREATE .DBS FILE FOR area-name

SCHEMA cannot create the DBS file for the specified area. Check that the file specification for the area is correct or that the protection for the file or directory allows creation. If the problem recurs, submit an SPR.

?DDLUIS UNABLE TO INITIALIZE SCHEMA FILE

SCHEMA cannot initialize the schema file. Check the file and its protection and try again. If the error recurs, submit an SPR.

?DDLVSA WHEN LOCATION MODE IS VIA-SET A SORTED MEMBER MUST BE AUTOMATIC

You did not specify that the member of a sorted set, which is also a via-set, is an automatic member. Make the membership automatic and try again.

?DDLVSN record-name's VIA-SET NEVER DEFINED IN SET ENTRY

You specified that the record was VIA a set, but you never defined that set in a set entry. Define the set or change the VIA phrase and try again.

?DDLWCD. WILDCARDING IN OUTPUT DIRECTORY

You cannot specify wildcard format in the output directory specification. Correct the command string and try again.

?DDLWNI. WILD-SPEC = NON-WILD-SPEC IS UNDEFINED

You cannot specify that an output specification has wild-card format and the input specification does not have wild-card format. Correct the file specification and try again.

%DDLWRR "WITHIN" NOT ASSOCIATED WITH A RECORD RANGE OF record-name

You specified a page range for a record in the DMCL area entry, but you did not include that area in a WITHIN clause for that record. Either move the page range to an area in a WITHIN clause or add the area to a WITHIN clause or let SCHEMA ignore the error.

?DDLXCT EXPLICIT TEXT AND COPY-TEXT CANNOT BE UNDER SAME DATA-NAME

You put text and the COPY phrase for that text with the same data-name. Move the text or the COPY phrase and try again.

%DDLXIS. EXTRA INPUT SPECS ARE IGNORED.

You included more than one input file specifications in the command line to SCHEMA. SCHEMA ignored all but the first.

%DDLXOS. EXTRA OUTPUT SPECS ARE IGNORED.

You included more than one output file specification in the command line to SCHEMA. SCHEMA ignored all but the first.

APPENDIX C
ORGANIZATION OF SCHEMA FILES

As described in Chapter 2, the SCHEMA program creates a schema file (SCH) as well as the data base files. The SCH file is structured like a data base file in that it contains pages and lines. It is thus itself a data base that describes the data base it controls. Each page in an SCH file is 512 words long including a 2-word header. Each line contains an occurrence of a record type that is SCHEMA-defined not user-defined. The record type IDs for these record types are shown in Table C-1. SCHEMA reserves ID numbers 001 through 032 (40_8) and assigns user-defined record type IDs starting with $033(41_8)$.

Table C-1
Record Type IDs for SCH Record Types

Record Type ID	Record Type
001	Schema Line
002	Record Line
003	Data Line
004	Control Line
005	Member Line
006	Owner Line
007	Within Line
010	Area Line
011	Text Line
012	Sub-Schema Line
013	Via Line
014	Item Line
015	File Line

The records in the SCH files describe the attributes of the data in the data base. These lines are used by DBCS to create the in-core representation of the data base. The set relationships among these lines are shown in Figure C-1.

The dotted arrows connecting the ITEM Line and the FILE Line to other lines indicate that a non-set relationship exists (i.e., these records are pointed to).

The lines (records) shown in Figure C-1 contain data that describe the attributes of the data base. These lines, with a detailed description of their contents, are described in alphabetical order in the following sections. Each line starts on a new page.

The lines dealing with record, set, area, and data-item names each contain a numeric field called a name ID. A name ID is an index into a table of these names. The name IDs for records, sets, and areas are passed to DBCS in place of the actual names.

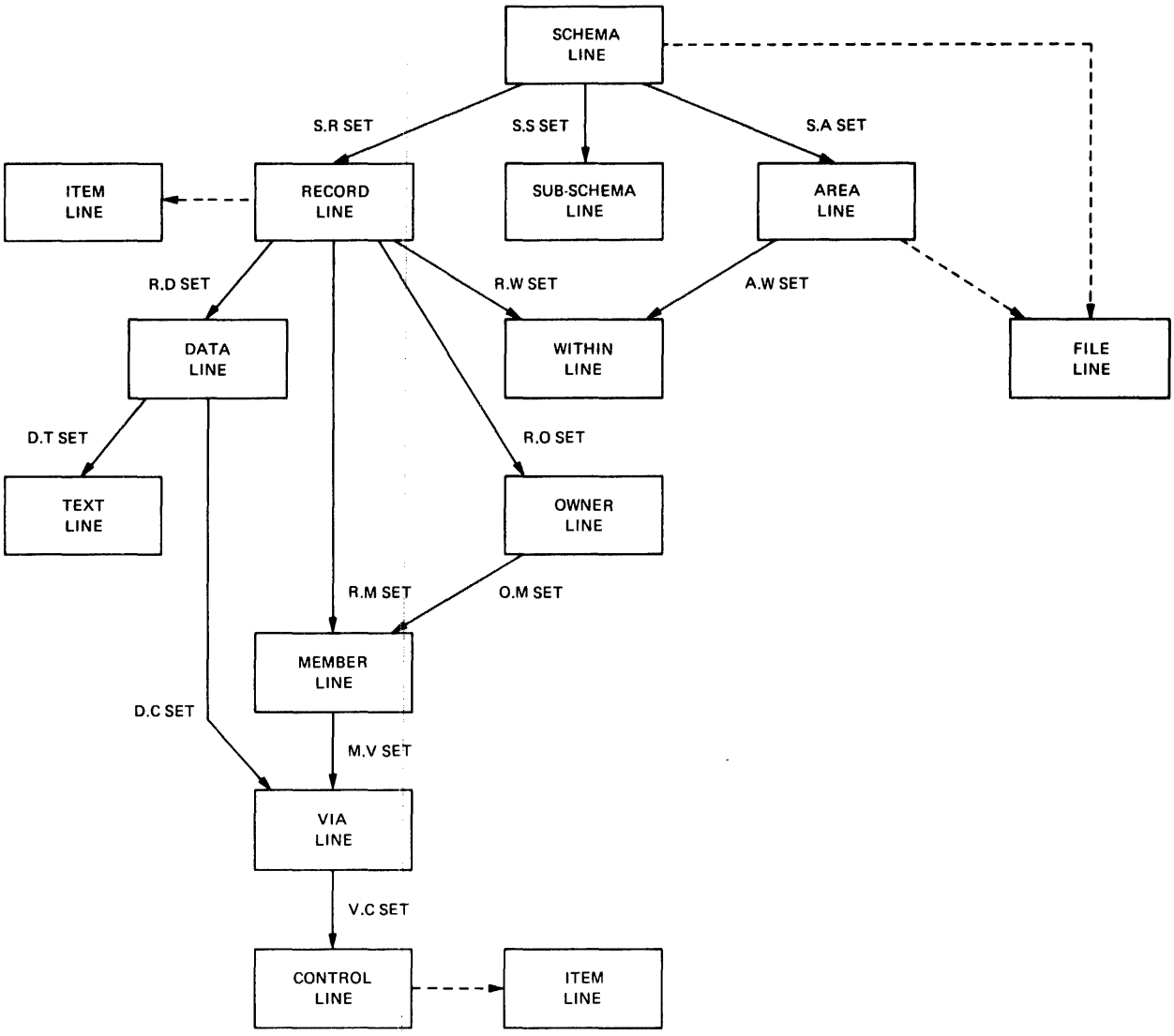


Figure C-1 Set Relationships in the Schema File

C.1 SCH AREA LINE

SCHEMA creates an AREA line for each area defined in the schema. The format of the SCH AREA line is shown in Figure C-2.

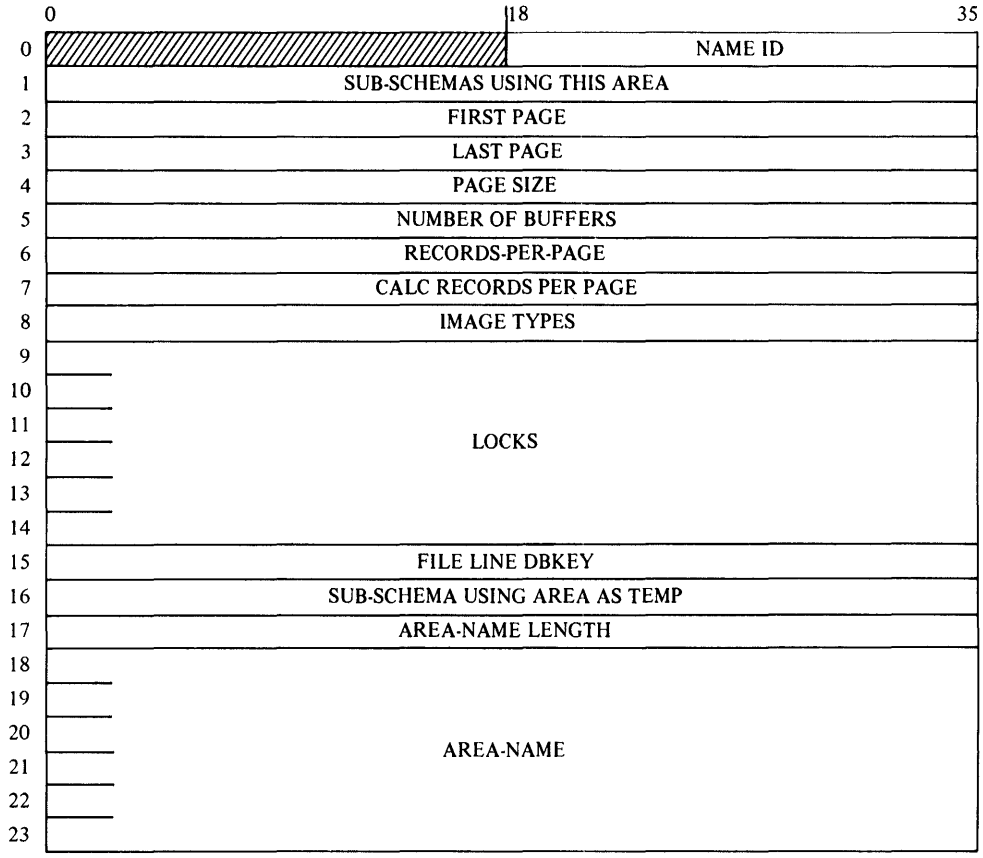


Figure C-2 SCH AREA Line

NAME ID contains the numeric identifier of the area-name.

SUB-SCHEMAS USING THIS AREA is a flag word each bit of which is set to 1 for each sub-schema in which the area is defined.

FIRST PAGE contains the number of the first page of the area.

LAST PAGE contains the number of the last page of the area.

PAGE SIZE contains the maximum number of words on a page.

NUMBER OF BUFFERS contains the number of buffers allocated for the area.

RECORDS-PER-PAGE contains the maximum number of records allowed on a page.

CALC RECORDS PER PAGE contains the number of CALC-chain header words allocated for each page of the area.

IMAGE TYPES describes the types of images to be written in the journal file. The values are:

0 - none
1 - AFTER images
2 - BEFORE images
3 - both BEFORE and AFTER images

LOCKS contains all the privacy locks for the area.

FILE LINE DBKEY contains the data base key of the FILE line for the area. The FILE line contains the file specification associated with the area.

SUB-SCHEMA USING AREA AS TEMP is a flag word each bit of which is set to 1 for each sub-schema in which this area is declared as temporary.

AREA-NAME LENGTH contains the length of the area-name.

AREA-NAME contains the name of the area in ASCII.

C.2 SCH CONTROL LINE

The SCH CONTROL line describes each of the subkeys in a sort key or a key used in set occurrence selection. The full key is described in a VIA line (see Section C.12). For each member in a sorted set or each member with a set occurrence selection of LOCATION MODE OF OWNER, SCHEMA creates as many VIA lines as there are keys. For each key, SCHEMA creates as many CONTROL lines as there are sub-keys in the key (or one if the key is not sub-divided). If a member record has neither sort keys nor a set occurrence selection key, that member has no VIA nor CONTROL lines associated with it. The format of a CONTROL line is shown in Figure C-3.

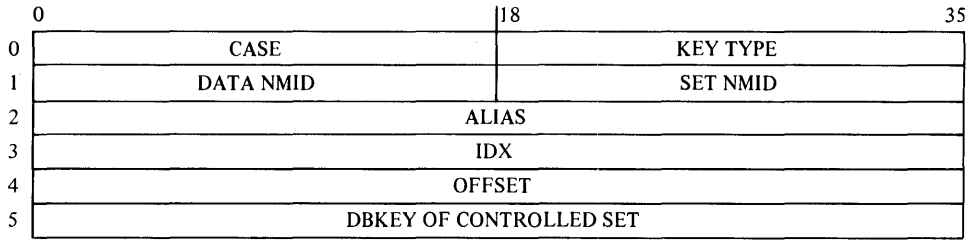


Figure C-3 SCH CONTROL Line

CASE contains one of the following values:

- 0 - the subkey is a data field
- 1 - the subkey is the database key of the actual record
- 2 - the subkey is the database key of the record as encoded in the index node
- 3 - the subkey is the record type ID of the record
- 4 - the subkey is the record type ID as encoded in the index node

KEY TYPE contains the following information:

- Bit 30 - the subkey is part of a sort key
- Bit 31 - the subkey controls a member not in the current sub-schema
- Bit 32 - the subkey is a CALC key
- Bit 33 - the subkey is a range key
- Bit 34 - the subkey is in descending sequence
- Bit 35 - the subkey is in ascending sequence

DATA NMID contains the name ID of the data-item used as the subkey.

SET NMID contains the name ID of the set controlled by the key.

ALIAS contains the database key of the ITEM line for the subkey's alias if one exists.

IDX contains the number of the subkey in the key (e.g , the first subkey is 1).

OFFSET contains the location of the key in the index node if the key is a sort key.

DBKEY OF CONTROLLED SET contains the database key of the set the key controls.

C.3 SCH DATA LINE

An occurrence of a DATA line is written for each 02-level data-name (whether data-item or data-aggregate) defined for the schema. This line (Figure C-4) is used to generate the in-core DATA block.

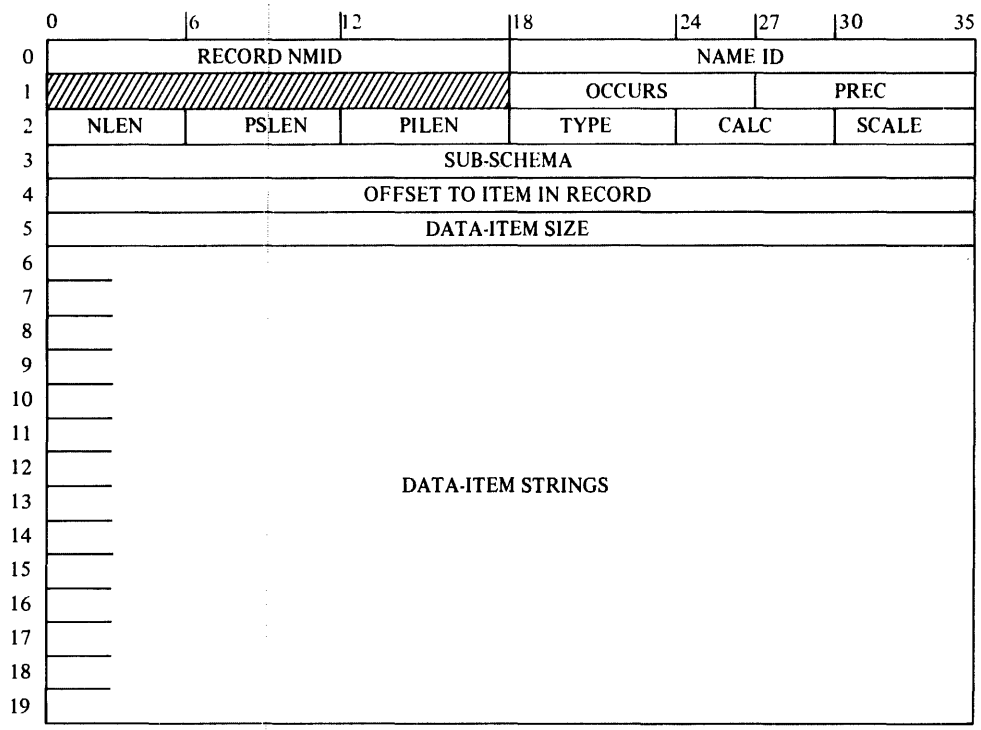


Figure C-4 SCH Data Line

RECORD NMID contains the numeric identifier of the record to which the data item belongs.

NAME ID contains the numeric identifier of the data-name.

OCCURS contains the number of times the item occurs as specified in an OCCURS clause.

PREC contains the precision of the item, e.g., S9(10) in COBOL.

NLEN contains the length of the data item's name.

PSLEN contains the length of the data item's pseudonym.

PILEN contains the length of the data item's PICTURE.

TYPE contains the data type of the item:

- 0 - no data type, i.e., the item is a data-aggregate
- 1 - the type is numeric, FIXED BINARY REAL
- 2 - the type is numeric, FLOAT BINARY REAL
- 3 - the type is numeric, FIXED DECIMAL REAL
- 4 - RESERVED
- 5 - RESERVED

6 - the TYPE is numeric, FLOAT BINARY COMPLEX
7 - RESERVED
8 - RESERVED
9 - the type is database key
10 - the type is alphanumeric, DISPLAY-6
11 - the type is alphanumeric, DISPLAY-7
12 - the type is alphanumeric, DISPLAY-9

CALC contains the number of the CALC field in the data-item.

SCALE contains the scale factor.

SUB-SCHEMA is a flag word each bit of which is set to 1 for each sub-schema that includes the data item.

POINTER TO ITEM IN RECORD contains the byte pointer to the actual position of the item in the record.

DATA-ITEM SIZE contains the size of the data-item.

DATA-ITEM is a variable-length field of up to 66 characters that contains the data-item's strings (in ASCII). These strings are the name of the data-item, its pseudonym if any, and its picture if any.

C.4 SCH FILE LINE

An SCH FILE line is created for each DBS file and for the journal file. It contains the file specification of the file and the file specification of the temporary file for a sub-schema temporary file. The FILE line format is shown in Figure C-5.

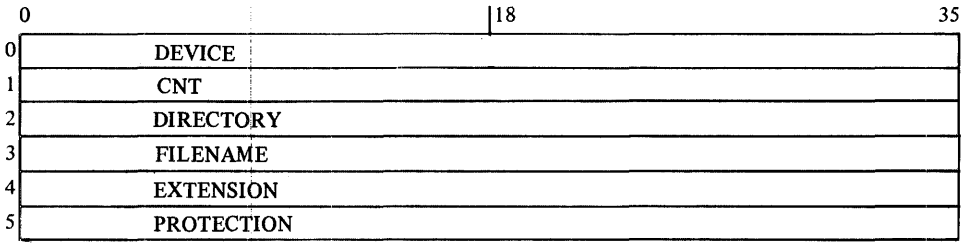


Figure C-5 SCH FILE Line

C.5 SCH ITEM LINE

SCHEMA creates an ITEM line for each record-independent name. These names are the DIRECT identifiers, the area-IDs, and the ALIAS identifiers. The format of an ITEM line is shown in Figure C-6.

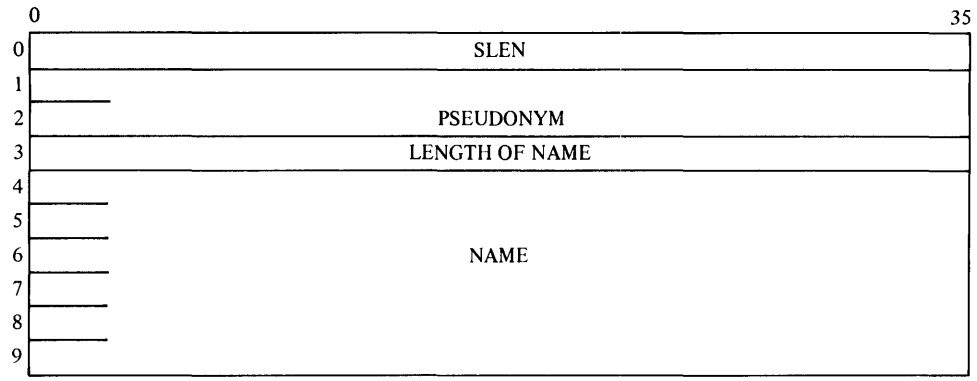


Figure C-6 SCH ITEM Line

SLEN contains the length of the name's pseudonym if one exists.

PSEUDONYM contains the pseudonym for the name (in ASCII) if you specified one. If there is no pseudonym, word 1 contains 0.

LENGTH OF NAME contains the number of characters in the name.

NAME contains the item's name (in ASCII). The name can be up to 30 characters long.

C.6 SCH MEMBER LINE

Each record that participates as a member of a set has associated with it a MEMBER line that identifies it as a member of that set. The SCH MEMBER line (Figure C-7) is used to form the in-core MEMBER block.

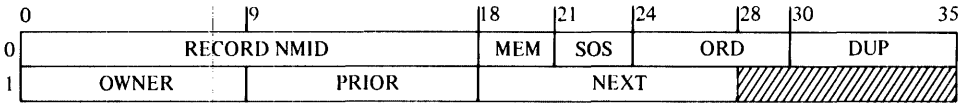


Figure C-7 SCH MEMBER Line

RECORD NMID contains the numeric identifier for the member record type.

MEM contains the type of membership - AUTOMATIC or MANUAL and MANDATORY or OPTIONAL.

SOS contains the form of set occurrence selection:

1 - CURRENT OF SET
2 - LOCATION MODE OF OWNER

ORD contains the set order:

1 - FIRST
2 - LAST
3 - NEXT
4 - PRIOR
5 - SORTED
6 - SORTED BY DATABASE-KEY
7 - SORTED WITHIN
8 - RESERVED
9 - SORTED WITH USER KEYS
10 - RESERVED
11 - SORTED WITHIN WITH USER KEYS
12 - SORTED WITH USER KEYS AND DUPLICATES ALLOWED

DUP refers to how duplicate keys are treated:

0 - duplicates are allowed
1 - duplicates are first
2 - duplicates are last
3 - duplicates are not allowed

OWNER contains the offset in the record of the OWNER pointer. This is 0 if you did not specify LINKED TO OWNER.

PRIOR contains the offset in the record of the PRIOR pointer. This is 0 if you did not specify LINKED TO PRIOR.

NEXT contains the offset in the record of the NEXT pointer.

C.7 SCH OWNER LINE

An OWNER line is written in the SCH file for each set type defined for the schema. The OWNER line is shown in Figure C-8 and is used as the basis for the in-core OWNER block.

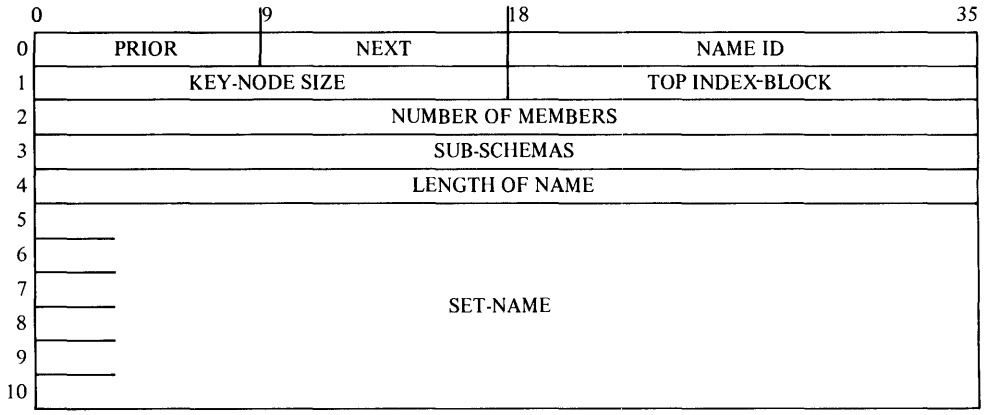


Figure C-8 SCH OWNER Line

PRIOR contains the offset in the owner record of its PRIOR pointer. If you did not specify LINKED TO PRIOR, it is 0 unless ORDER IS LAST.

NEXT contains the offset in the owner record of its NEXT pointer.

NAME ID contains the numeric identifier of the set-name.

KEY-NODE SIZE contains the largest index-node size for members of this set.

TOP INDEX-BLOCK contains the offset of the pointer to the top index-block.

NUMBER OF MEMBERS contains the number of member records in the set.

SUB-SCHEMAS is a flag word each bit of which is set to 1 for each sub-schema in which the set is defined.

LENGTH OF NAME is the length of the set-name.

SET-NAME contains the name of the set (in ASCII). It can be up to 30 characters long.

C.8 SCH RECORD LINE

An SCH RECORD line is created for each record; it identifies a record according to its type, name, and location. Figure C-9 shows the format of the SCH RECORD line.

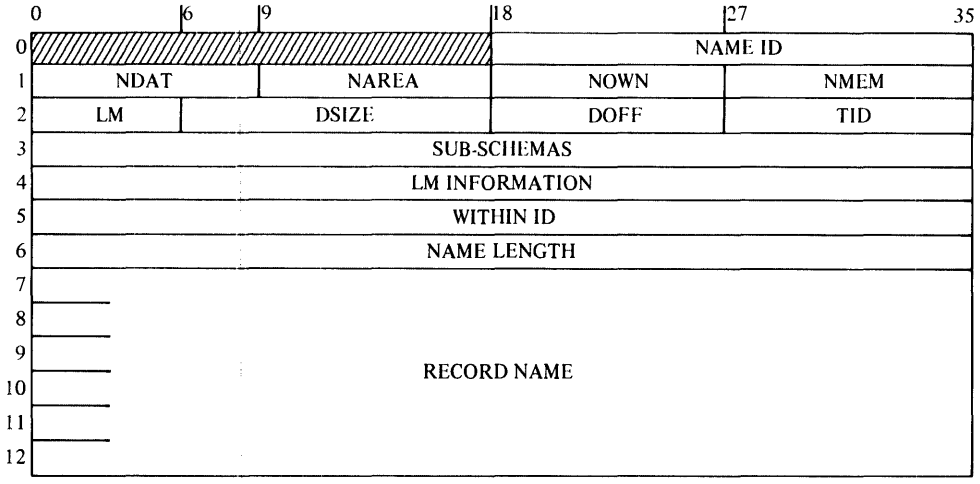


Figure C-9 SCH RECORD Line

NAME ID contains the numeric identifier of the record-name.

NDAT contains the number of data-items that are part of this record.

NAREA contains the number of areas in which this record can reside.

NOWNER contains the number of sets this record owns.

NMEM contains the number of sets in which this record is a member.

LM contains the location mode:

- 0 - none (only applies to the system record)
- 1 - DIRECT
- 2 - VIA
- 3 - CALC with duplicates allowed
- 4 - CALC with duplicates not allowed

DSIZE contains the size of the data in words.

DOFF contains the offset in the record of the first word of the data.

TID contains the record type ID of the record type.

SUB-SCHEMAS is a flag word each bit of which is set to 1 for each sub-schema in which the record is defined.

LM INFORMATION contains information that depends on the location mode:

- If it is CALC - number of CALC fields.
- If it is VIA - database key of the OWNER line of the set named in the VIA phrase.
- If it is DIRECT - database key of the ITEM line that describes the DIRECT identifier.

WITHIN ID contains the database key of the ITEM line for the area ID, if one exists.

NAME LENGTH contains the length of the record-name.

RECORD NAME contains the name of the record (in ASCII). The name can be up to 30 characters long.

C.9 SCH SCHEMA LINE

The SCH SCHEMA line contains information about the schema such as the number of sub-schemas, number of areas, and the journal's FILE line. It is used to create the in-core SCHEMA block. The format of the SCH SCHEMA line is shown in Figure C-10.

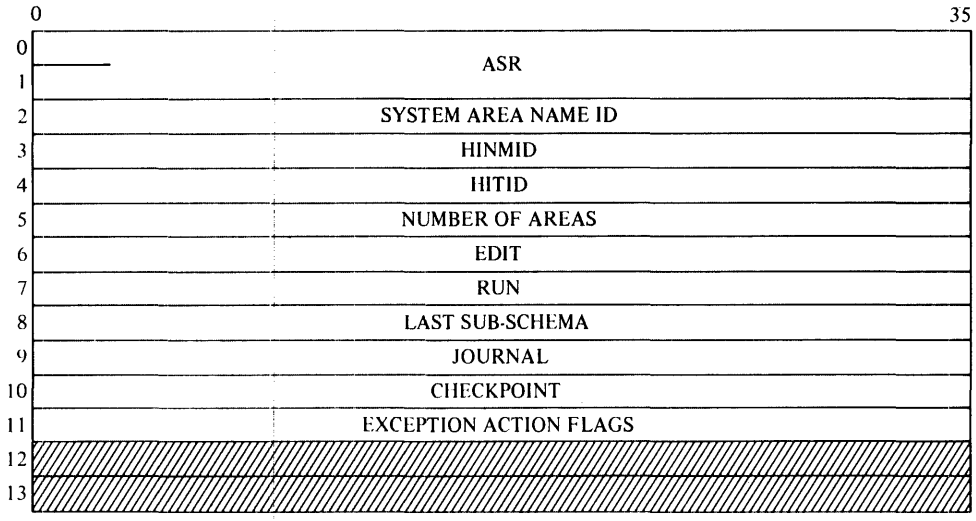


Figure C-10 SCH SCHEMA Line

ASR contains an Area Status Record like the one in a DBS file.

SYSTEM AREA NAME ID contains the numeric identifier of the area that contains the system record.

HINMID contains the number that is the highest name ID used.

HITID contains the number that is the highest record type ID used.

NUMBER OF AREAS contains the number of areas defined for the schema.

EDIT contains the number of the last edit to the schema.

RUN contains the number of times the schema has been used by run-units.

LAST SUB-SCHEMA contains the number of the last sub-schema declared. The highest possible value is 36.

JOURNAL contains the database key of the journal's FILE line.

CHECKPOINT contains checkpointing flags:

- bit 34 - VER flag (checkpointing by command)
- bit 35 - TR flag (checkpointing by transaction)

EXCEPTION ACTION FLAGS is two half words of flags. The left half is for NOTE exceptions; the right half is for INTERCEPT exceptions. The relevant bits are:

- bits 13 and 30 - SYS flag (system exceptions will be noted/intercepted)
- bits 14 and 31 - ALL flag (all exceptions will be noted/intercepted)
- bits 15 and 33 - BIND flag (exceptions during binding will be noted/intercepted)
- bits 16 and 34 - CALL flag (exceptions during calls will be noted/intercepted)
- bits 17 and 35 - UPD flag (exceptions during updating will be noted/intercepted)

C.10 SCH SUB-SCHEMA LINE

An SCH SUB-SCHEMA line is created for each sub-schema using the schema. Its format is shown in Figure C-11.

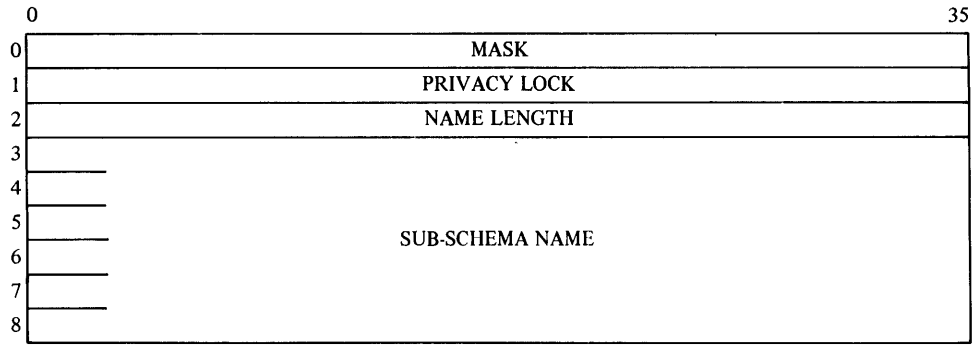


Figure C-11 SCH SUB-SCHEMA Line

MASK contains a mask that is ORed with other SCH lines. If the result is non-zero, the line is used in the sub-schema.

PRIVACY LOCK contains the privacy lock for the sub-schema if one exists.

NAME LENGTH contains the length of the sub-schema name.

SUB-SCHEMA NAME contains the sub-schema name in ASCII. The name can be up to 30 characters long.

C.11 SCH TEXT LINE

One or more TEXT lines are created for each data-aggregate in the schema. Figure C-12 shows the format of an SCH TEXT line.

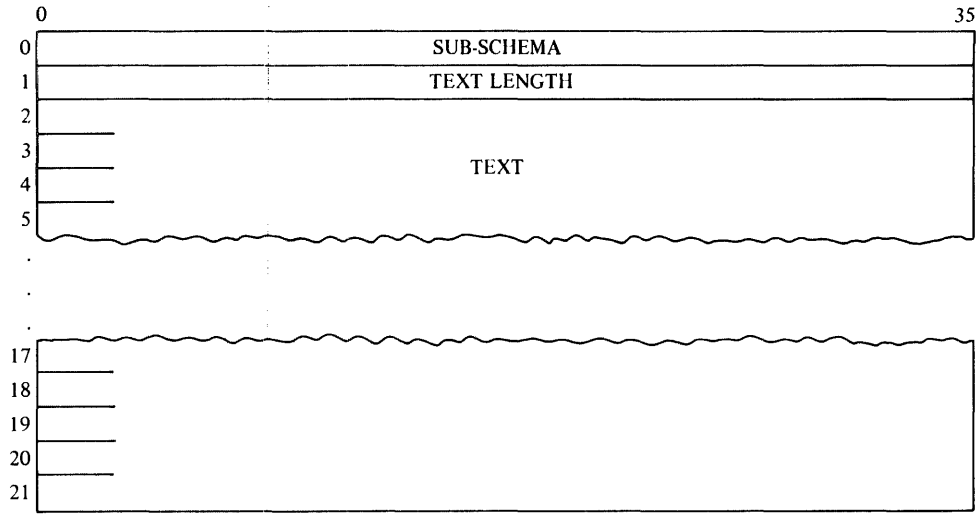


Figure C-12 SCH TEXT Line

SUB-SCHEMA contains the numbers of the sub-schema in which the text is defined. Each bit of the word represents a sub-schema.

TEXT LENGTH contains the number of characters in the text.

TEXT contains the text in ASCII. The text can be up to 100 characters long.

C.12 SCH VIA LINE

A SCH VIA line is created for each member that has a sort control key or a set occurrence selection key. If the member has both keys, a VIA line is created for each. The VIA line points to the CONTROL lines for the sub-keys in the key. The SCH VIA line is one word long and is shown in Figure C-13.

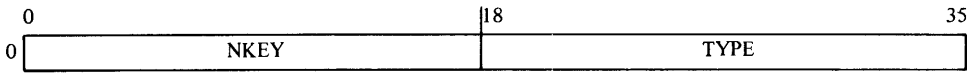


Figure C-13 SCH VIA Line

NKEY contains the numbers of subkeys in each key.

TYPE contains the type of the key:

1 - DIRECT key used in set occurrence selection
2 - CALC key used in set occurrence selection
3 - VIA key used in set occurrence selection
4 - SORTED key used in sort control

C.13 SCH WITHIN LINE

An SCH WITHIN line is created for each record that can be in each area. It contains the record-type and its page ranges. Figure C-14 illustrates this line.

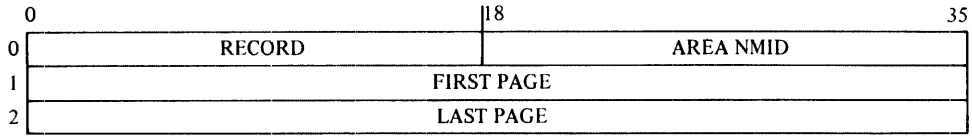


Figure C-14 SCH WITHIN Line

RECORD contains -1 if the record type appeared in a page range clause for the area. If it does not, RECORD contains 0.

AREA NMID contains the numeric identifier of the area to which this line is connected.

FIRST PAGE contains the number of the first page of the range specified for the record. If you did not specify a range, the first page of the area is used.

LAST PAGE contains the number of the last page of the range specified for the record. If you did not specify a range, the last page of the area is used.

APPENDIX D
DATA ORGANIZATION AND ACCESS

As stated in Chapter 2, SCHEMA creates a DBS file for each area in the schema. The data base is composed of these DBS files. This appendix describes the format of a page in a DBS file, the in-core representation of the data base, and the overhead involved in accessing the DBS files.

D.1 FORMAT OF A PAGE

A page in a DBS file contains a page header and lines containing records. The format of a page header is shown in Figure D-1.

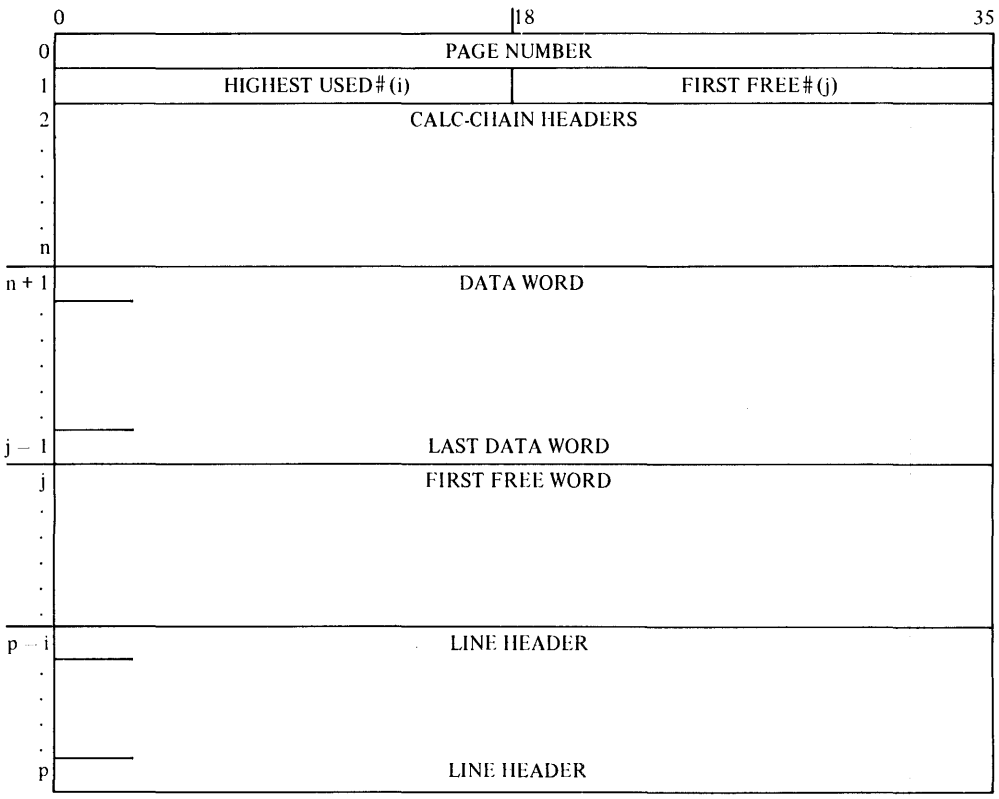


Figure D-1 Format of a Page Header

PAGE NUMBER contains the number of this page.

HIGHEST USED contains the highest line number used so far on this page.

FIRST FREE contains the offset of the first word on the page on which tnere is no data.

CALC CHAIN HEADERS is 0 or more words (depending on the value specified in the CALC RPP clause). Each word is the database key of the first record in each (non-empty) CALC chain.

Each line on a page begins and ends on a word boundary and covers as many words as necessary to contain the data. Lines cannot cross page boundaries. Each line contains a line header, set pointers, and the data. The format of a line header is shown in Figure D-2.

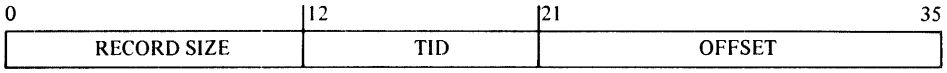


Figure D-2 Format of a Line Header

RECORD SIZE contains the size of the record occurrence on the line.

TID contains the record type ID of the record occurrence.

OFFSET contains the offset from the beginning of the page to the first word of data in the record.

For each set link associated with the record occurrence on a line, DBCS includes a set pointer on the line. Each pointer is one word, which is the database key of the line to which it points. A database key is the combination of the number of the line and the number of the page on which a record resides. Figure D-3 shows the format of a database key.

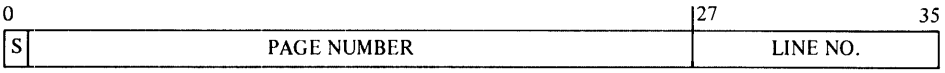


Figure D-3 Format of a Database Key

S is the sign of the key.

PAGE NUMBER contains the page number.

LINE NO contains the line number.

The first line of the first page in an area is the Area Status Record. Its format is shown in Figure D-4.

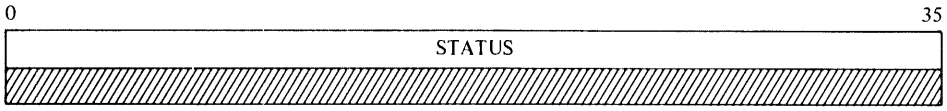


Figure D-4 Format of an Area Status Record

STATUS contains the status of the area. If the area is in a state of flux (i.e., opened for update), this word is set to -1. If the area is open and the run-unit accessing it aborts, the -1 indicates that the area is in an undefined state. If the area is closed and available for normal use, this word is set to 0.

D.2 IN-CORE BLOCKS

When an application program invokes a sub-schema, DBCS builds the in-core blocks for the data base according to the SCH lines for the schema. These in-core blocks contain the descriptions of the records, sets, and areas in the sub-schema. Each block is described below on a separate page.

Several of the blocks contain AOBJ pointers that point to tables. The tables consist of single-word entries. Each table has its own format, but all the AOBJ pointers have the same format. The left half contains a negative count of the entries in the table and the right half contains the pointer to the table.

D.2.1 In-Core AREA Block

An AREA block is written in memory for each area known to the invoked sub-schema. Its format appears in Figure D-5.

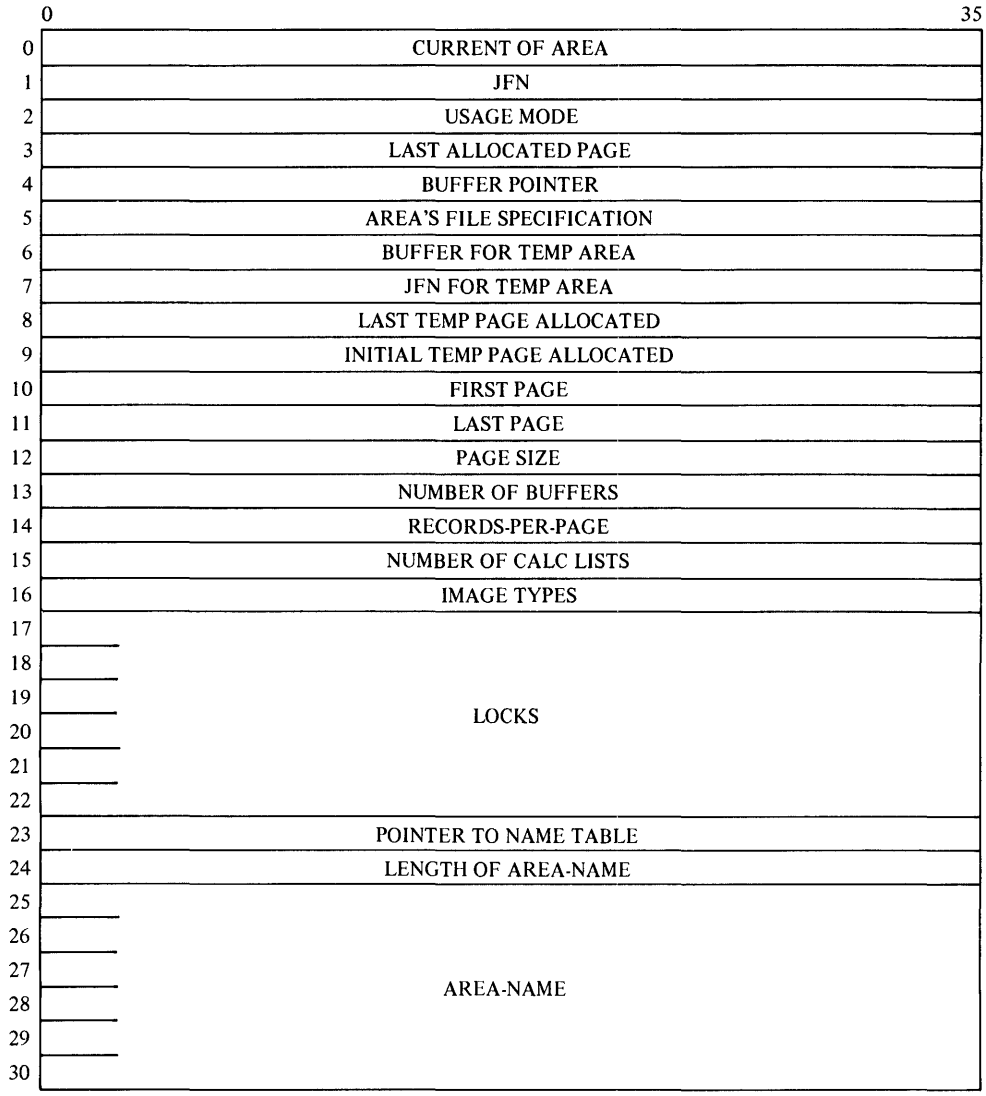


Figure D-5 In-Core AREA Block

CURRENT OF AREA contains the database key of the current record occurrence of this area.

JFN contains the JFN of the file to which this area belongs.

USAGE MODE refers to the USAGE MODE specified for the OPEN statement. The values and the modes they represent are:

0 - RETRIEVAL
1 - UPDATE
2 - PROTECTED RETRIEVAL
3 - PROTECTED UPDATE
4 - EXCLUSIVE RETRIEVAL
5 - EXCLUSIVE UPDATE

LAST ALLOCATED PAGE contains the number of the last page allocated to the area.

BUFFER POINTER contains an AOBJ pointer to the list of buffer pointers for this area. The list of buffer pointers contains as many words as there are buffers for this area. The right half of each word contains the pointer to each buffer.

AREA'S FILE SPECIFICATION contains the pointer to the FILE block for the area.

BUFFER FOR TEMP AREA contains the address of an extra buffer allocated for the temporary area's directory.

JFN FOR TEMP AREA contains an additional JFN needed if the area is opened as temporary.

LAST TEMP PAGE ALLOCATED contains the number of the last page allocated in the temporary area.

INITIAL TEMP PAGE ALLOCATED contains the number of the initial last allocated page in the data base file.

FIRST PAGE is the number of the first page in the area.

LAST PAGE contains the number of the last defined page in the area.

PAGE SIZE contains the number of words in a page.

NUMBER OF BUFFERS contains the number of buffers that are allocated for this area.

RECORDS-PER-PAGE contains the maximum number of records that can be stored on a page.

NUMBER OF CALC-CHAINS contains the number of CALC-chains that are allowed on each page.

IMAGE TYPES contains the type of page images being written in the journal file. The value can be one of:

0 - none
1 - AFTER images
2 - BEFORE images
3 - both BEFORE and AFTER images

LOCKS contains the text of all the privacy locks declared for this area.

POINTER TO NAME TABLE contains a pointer to the list of area name IDs.

LENGTH OF AREA NAME contains the length of the area-name.

AREA-NAME contains the area-name (in ASCII). It is a variable-length field up to 30 characters long.

D.2.2 In-Core DATA Block

An in-core DATA Block is created for each 02 data-name belonging to a record in an invoked sub-schema. The DATA block is created from the SCH DATA line. The format of a DATA block is shown in Figure D-6.

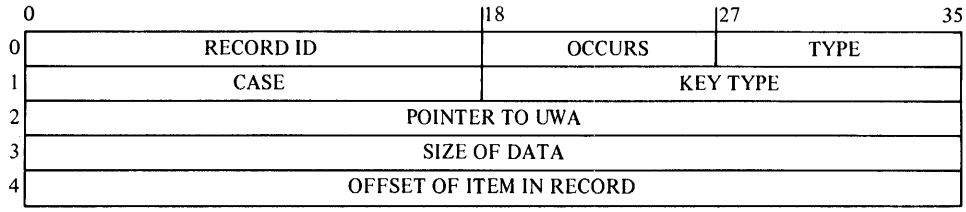


Figure D-6 In-Core DATA Block

RECORD ID contains the record type ID of the record in which the data-item belongs.

OCCURS contains the number of times the data-item occurs as specified in an OCCURS clause.

TYPE contains the data type of the item:

- 0 - no data type, i.e., the item is a data-aggregate
- 1 - the type is numeric, FIXED BINARY REAL
- 2 - the type is numeric, FLOAT BINARY REAL
- 3 - the type is numeric, FIXED DECIMAL REAL
- 4 - RESERVED
- 5 - RESERVED
- 6 - the type is numeric, FLOAT BINARY COMPLEX
- 7 - RESERVED
- 8 - RESERVED
- 9 - the type is database key
- 10 - the type is alphanumeric, DISPLAY-6
- 11 - the type is alphanumeric, DISPLAY-7
- 12 - the type is alphanumeric, DISPLAY-9

CASE contains one of the following values:

- 0 - the data-item is not a key (sort or set occurrence selection)
- 1 - the database key of the actual record is its key
- 2 - the database key of the record as encoded in the index list is the key
- 3 - the record type ID of the record is the key
- 4 - the record type ID as encoded in the index list is the key

KEY TYPE contains one of the following values:

- Bit 30 - the data-item is the entire key
- Bit 31 - the key controls a member not in the current sub-schema
- Bit 32 - the data-item is a CALC key
- Bit 33 - the data item is a range key
- Bit 34 - the key is in descending sequence
- Bit 35 - the key is in ascending sequence

Word 1 (CASE and KEY TYPE) is set to 0 if the data-item is not a key.

POINTER TO UWA contains a byte pointer to the data-item as it exists in the UWA.

SIZE OF DATA contains the size of the data-item in the applicable unit (e.g., characters, words, depending on the data-type). This number is the product of the size of the data-item times the number of times it occurs.

OFFSET OF ITEM IN RECORD contains a byte pointer to the data item's offset in the record.

D.2.3 In-Core FILE Block

A FILE block is created in memory for each DBS file and for the journal file. It is created from the SCH FILE line. The format of a FILE block is shown in Figure D-7.

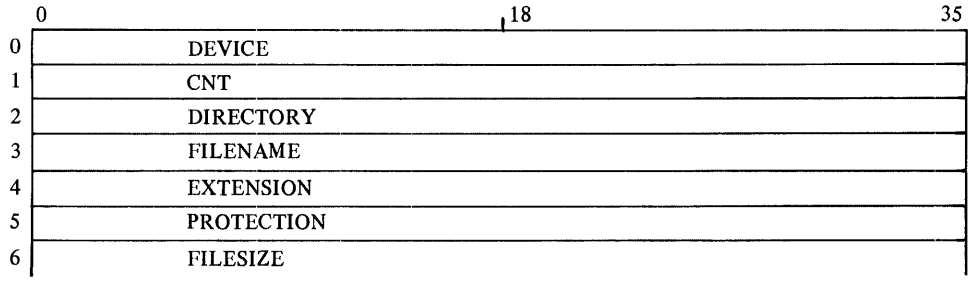


Figure D-7 In-Core FILE Block

D.2.4 In-Core MEMBER Block

A MEMBER block is created from the SCH MEMBER line for each member record in every set. Figure D-8 shows the format of a MEMBER block.

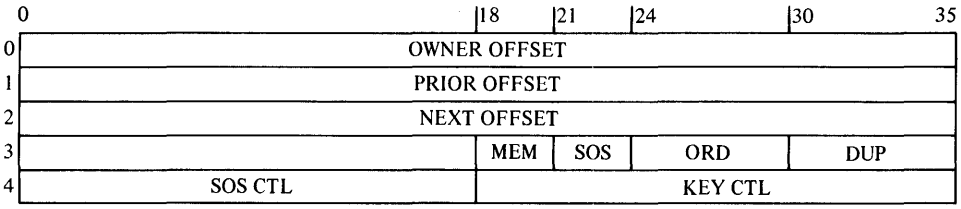


Figure D-8 In-Core MEMBER Block

OWNER OFFSET contains the offset in the record of the OWNER pointer. This is 0 if you did not specify LINKED TO OWNER.

PRIOR OFFSET contains the offset in the record of the PRIOR pointer. This is 0 if you did not specify LINKED TO PRIOR.

NEXT OFFSET contains the offset in the record of the NEXT pointer.

MEM contains the type of membership - AUTOMATIC or MANUAL and MANDATORY or OPTIONAL.

SOS contains the form of set occurrence selection:

1 - CURRENT or SET
2 - LOCATION MODE OF OWNER

ORD contains the set order:

1 - FIRST
2 - LAST
3 - NEXT
4 - PRIOR
5 - SORTED
6 - SORTED BY DATABASE-KEY
7 - SORTED WITHIN
8 - RESERVED
9 - SORTED WITH USER KEYS
10 - RESERVED
11 - SORTED WITHIN WITH USER KEYS
12 - SORTED WITH USER KEYS AND DUPLICATES ALLOWED

DUP contains the description of how duplicate keys are treated:

0 - duplicates are allowed
1 - duplicates are first
2 - duplicates are last
3 - duplicates are not allowed

SOS CTL contains a pointer to the VIA block for the set occurrence selection key.

KEY CTL contains pointers to the VIA blocks for the sort control keys.

D.2.5 In-Core OWNER Block

An in-core OWNER block is created from the SCH OWNER line for each set referenced in the sub-schema. An OWNER block is variable in length and its format is shown in Figure D-9.

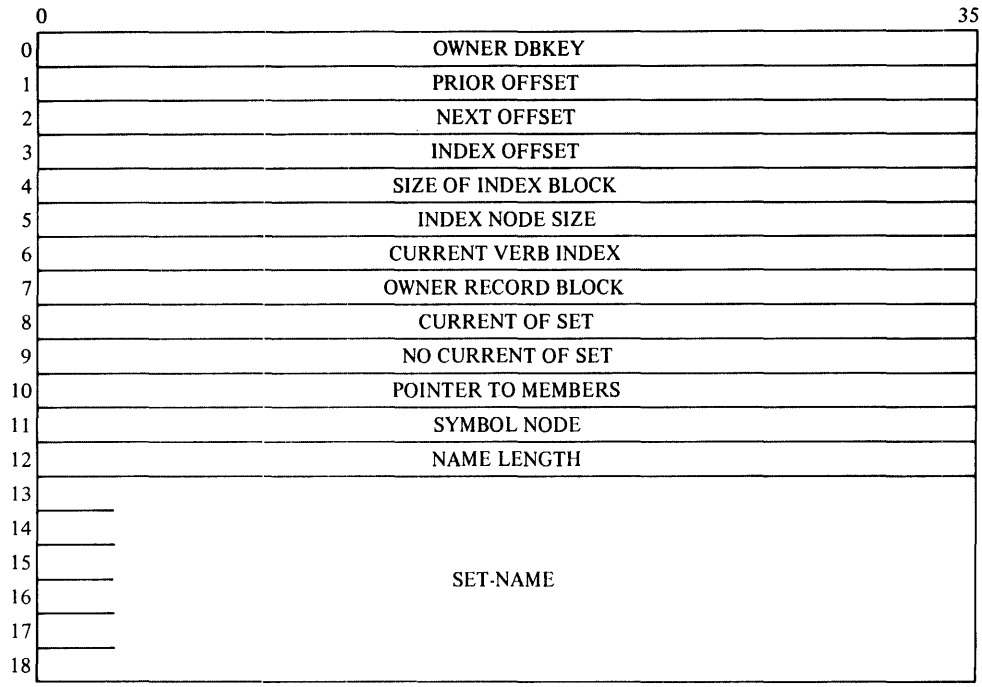


Figure D-9 In-Core OWNER Block

OWNER DBKEY contains the database key of the owner of the current set occurrence.

PRIOR OFFSET contains the offset in the owner record of its PRIOR pointer. If you did not specify LINKED TO PRIOR, it is 0.

NEXT OFFSET contains the offset in the owner record of its NEXT pointer.

INDEX OFFSET contains the offset to the pointer to the head of an index structure if the set is sorted.

SIZE OF INDEX BLOCK contains the number of words in the index block for the set.

INDEX NODE SIZE contains the size of a node in the index block.

CURRENT VERB INDEX is for error recovery purposes.

OWNER RECORD BLOCK contains the address of the RECORD block for the owner record of the set.

CURRENT OF SET contains the database key of the current record of the set.

NO CURRENT OF SET contains the canonical next of set if the current record was deleted or removed from the set.

POINTER TO MEMBERS contains an AOBJ pointer to a table of member blocks. Each word of the table contains a pointer to the RECORD block of each member and a pointer to the MEMBER block of each member.

SYMBOL NODE contains a pointer to the symbol node.

NAME LENGTH contains the length of the set-name.

SET NAME contains the set-name (in ASCII). It is a variable-length field up to 30 characters long.

D.2.6 In-Core RECORD Block

A RECORD block is created in memory for each record type used in the sub-schema. The in-core RECORD block is derived from the SCH RECORD line. The format of an in-core RECORD block is shown in Figure D-10.

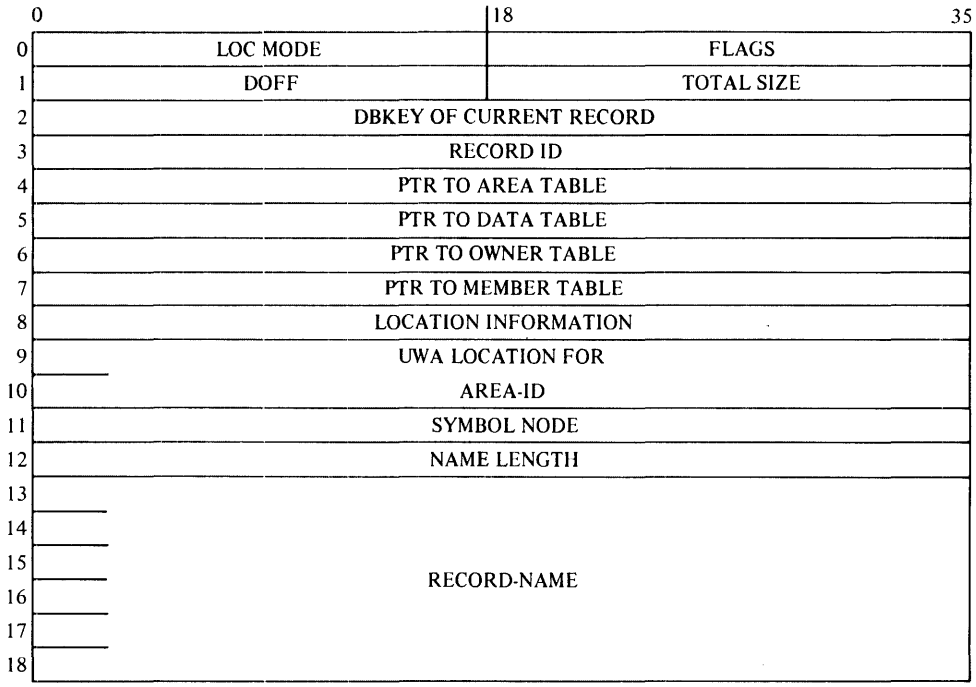


Figure D-10 In-Core RECORD Block

LOC MODE contains the location mode:

- 0 - none, i.e., the record is the system record
- 1 - DIRECT
- 2 - CALC
- 3 - VIA

FLAGS contains the following:

- Bit 34 - record cannot be deleted because a set in which it belongs is not in the sub-schema.
- Bit 35 - record cannot be stored/deleted because the set owned by the record is not in the sub-schema.

DOFF contains the data offset.

TOTAL SIZE contains the total number of words in the record.

DBKEY OF CURRENT RECORD contains the database key of the current occurrence of this record type.

RECORD ID contains the record type ID of the record.

PTR TO AREA TABLE contains an AOBJ pointer to a table. Each word in the table contains two pointers. The pointer in the right half points to an area in which the record can reside. The pointer in the left half points to the WITHIN block associated with this area/record pair.

PTR TO DATA TABLE contains an AOBJ pointer to a table. Each word in the table contains a pointer in the right half that points to each DATA block associated with the record.

PTR TO OWNER TABLE contains an AOBJ pointer to a table. Each word in the table contains a pointer in the right half to the owner block for the record.

PTR TO MEMBER TABLE contains an AOBJ pointer to a table. Each word in the table contains two pointers. The pointer in the left half points to the MEMBER blocks for the record. The pointer in the right half points to the OWNER blocks of the sets in which the record is a member.

LOCATION INFORMATION contains information that depends on the location mode:

If it is CALC	-	AOBJ pointer to a table of CALC keys.
If it is VIA	-	database key of the OWNER block of the set named in the VIA phrase.
If it is DIRECT	-	UWA address of the DIRECT identifier.

UWA LOCATION FOR AREA-ID contains a string pointer to the UWA location for the area-ID if one exists.

SYMBOL NODE contains a pointer to the symbol node that contains the record-name.

NAME LENGTH contains the length of the record-name.

RECORD-NAME contains the record-name in ASCII. It is a variable-length field up to 30 characters long.

D.2.7 In-Core VIA Block

A VIA block is created in memory for each sort key or set occurrence selection key. It is created from the SCH VIA line. The format of an in-core VIA block is shown in Figure D-11.

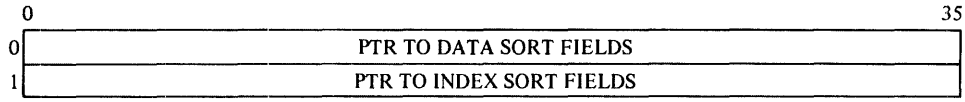


Figure D-11 In-Core VIA Block

PTR TO DATA SORT FIELDS contains an AOBJ pointer to a table. Each word in the table contains two values. The value in the left half is the type of the key. The value in the right half is a pointer to the key in the record.

PTR TO INDEX SORT FIELDS contains an AOBJ pointer to a table. Each word in the table contains two values. The value in the left half is the type of the key and the value in the right half is a pointer to the key in the index node.

D.2.8 In-Core WITHIN Block

For each area in which a record can reside, a WITHIN block is created in memory. Figure D-12 shows this block.

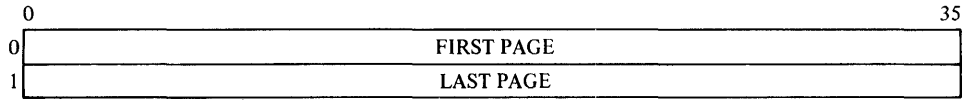


Figure D-12 In-Core WITHIN Block

FIRST PAGE contains the number of the first page of the range of the record in the area. If you did not specify a range, the first page of the area is used.

LAST PAGE contains the number of the last page of the range of the record in the area. If you did not specify a range, the last page of the area is used.

D.3 OVERHEAD

DBMS requires a certain amount of overhead to hold in memory such information as the linkages, the page headers, and the line headers. You can estimate overhead in terms of records (lines), pages, files, and run-units.

D.3.1 Record Overhead

For each record type defined for the data base, you can use the following to determine the amount of overhead in words for each record occurrence:

1. location mode is CALC	1 word
2. owner of set types	1 word per set
3. member of set types	1 word per set
4. LINKED TO OWNER in set types	1 word per set
5. LINKED TO PRIOR in set types	1 word per set

Add to the calculation 1 word for the line header, and you have found the overhead for each occurrence of this record type. Consider the following examples:

EXAMPLE 1

The record type INVENTORY-RECORD is a CALC record, is the owner of 4 sets, participates as a member in 5 sets, is LINKED TO OWNER in 3 of these, and is LINKED TO PRIOR in one of them. Using the above overhead determination you can calculate the record overhead as:

1	(for CALC chain)
4	(for owner of 4 sets)
5	(for member of 5 sets)
3	(for OWNER links)
1	(for PRIOR link)
14	

Adding the word for the line header gives a total of 15 words of overhead for each occurrence of this record type.

EXAMPLE 2

The record type SUPPLIER-RECORD is a DIRECT record, and owns only one set. The line (record) overhead for this record type would be one word for the line header and one word for being an owner. Thus each line on which this record type appeared would have 2 words of overhead.

A stand-alone record - - i.e., one with no set linkages - - will have a minimum of two words of overhead.

There will be times when you will have to decide whether to repeat a certain item of data in more than one record or to create a set to eliminate the data redundancy. If the redundant data would take up less than three or four words in storage, the justification of set-link overhead would be questionable. That is, questionable from the point of view of efficient use of storage. This has to be traded off against the undesirable aspect of having to update two or more records in the data base.

D.3.2 Page Overhead

Page overhead consists of the number of words that compose the page header. In general, this overhead can be calculated using the formula:

Page Overhead = 2 + number of CALC chains

The 2 is for the two fixed words in the page header. In Example 1, the page overhead would be 3 words; in Example 2, it would be 2 words.

D.3.3 File Overhead

In addition to page and line overhead, every DBS file has a three-word sector on the first page of the file used to store the Area Status Record. This overhead is inherent to the system and cannot be altered.

If you have specified OWNER IS SYSTEM at all, overhead for the file will be increased by the number of pointers in the system set.

D.3.4 Run-unit Overhead

For each run-unit, overhead depends on the number of sub-schemas invoked plus a fixed amount for DBCS. The overhead consists of:

1. A DBCS internal control area (currently about 200 words).
2. An in-core data base (see Section D.2) for each sub-schema. The size of this data base depends on the number of blocks needed for the sub-schema.
3. A UWA for each sub-schema.
4. Buffers for each area opened at some time during execution of the run-unit. If an area is temporary, an additional buffer is required for that area.

D.4 STORE ALGORITHM

As described in Section 2.2.3.2, DBCS uses the location mode specified for a record to locate the approximate page on which to store a record. Once DBCS locates the approximate page, it stores a record in the same manner regardless of its location mode. That is, the location mode only specifies the algorithm that DBCS uses to find the page on which to try to store the record occurrence. It tries to store a record occurrence on the first empty line on that page. If there is no more room on the page, DBCS tries to store the record on the next page. If there is no room on that page, DBCS calculates the next page on which to look for room by adding a prime number to the page number of the last page it checked. It continues to do this (cycling back through the area from its beginning if necessary) until either room is found for the record or all pages have been checked and there is no room, in which case DBCS returns an exception code.

When DBCS finds an empty line for the record occurrence, it sets the record's pointers and, in the case of a CALC record, adds the record to the CALC-chain on the page it originally calculated.

INDEX

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

DECsystem-10
Data Base Management
System Administrator's
Procedures Manual
AA-0899C-TB

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Performance Report (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 cut along this line.

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 _____

If you require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 152
MARLBORO, MASS.

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

Postage will be paid by:

digital

Software Documentation
200 Forest Avenue MR1-2/E37
Marlboro, Massachusetts 01752