VMS Software

VSI OpenVMS

VSI COBOL for OpenVMS DBMS Database Programming Manual

Operating System and Version: VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher

Software Version: VSI COBOL Version 3.1-7 for OpenVMS

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

VSI COBOL for OpenVMS DBMS Database Programming Manual

VMS Software

Legal Notice

Table of Contents

Preface

This manual describes how to develop Oracle CODASYL DBMS database programs with VSI COBOL for OpenVMS on the OpenVMS Alpha, OpenVMS Industry Standard 64 (OpenVMS I64), or OpenVMS VAX operating system. It also contains information about VSI COBOL for OpenVMS language features specific to OpenVMS.

The *VSI COBOL DBMS Database Programming Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-dbms-database-programming-manual/] is a component of the VSI COBOL for OpenVMS documentation set. Complete information about VSI COBOL for OpenVMS can be found in the *VSI COBOL User Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-user-guide/] and *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/].

VSI COBOL for OpenVMS is a VSI implementation of COBOL (COmmon Business-Oriented Language), which is widely used throughout the world for business data processing. VSI COBOL for OpenVMS is a high-performance language for commercial application development that runs under the OpenVMS Alpha, OpenVMS I64, or OpenVMS VAX operating system. VSI COBOL for OpenVMS is based on the *1985 ANSI COBOL Standard X3.23–1985* and *Federal Information Processing Standard Publication 21-3* (FIPS-PUB 21-3).

The FIPS standard identifies the ANSI standard as the standard adopted by the U.S. federal government and as the criteria on which federal validation is based. VSI COBOL for OpenVMS also contains VSI extensions to COBOL.

Oracle CODASYL DBMS, including DML (data manipulation language), is based on the 1979 CODASYL Standard for databases.

VSI COBOL is the new name for what has formerly been known as Compaq COBOL, DEC COBOL, DIGITAL COBOL, and VAX COBOL. VSI COBOL, unmodified, refers to the following products:

VSI COBOL for OpenVMS Industry Standard 64
VSI COBOL for OpenVMS Alpha
VSI COBOL for UNIX
VSI COBOL for OpenVMS VAX

Any references to the former names in product documentation or other components should be construed as references to the VSI COBOL names.

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. Intended Audience

This manual is designed for experienced applications programmers who have a basic understanding of the usage of the COBOL language and database programming. If you are a new COBOL user, you may need to read introductory COBOL textbooks or take COBOL courses. Some familiarity with your operating system is also recommended. This is not a tutorial manual.

3. Document Structure

This manual is organized as follows:

- Chapter 1, describes the structure and organization of a VSI COBOL for OpenVMS database program, and explains how you compile, link, and run your program.
- Chapter 2, describes elements of the VSI COBOL for OpenVMS language that support database programming.
- Chapter 3, describes logical and physical concepts of the Data Division in relation to database programs.
- Chapter 4, describes statements, names, rules, and conditions used in the Procedure Division of a VSI COBOL for OpenVMS database program.
- Chapter 5, describes VSI COBOL for OpenVMS for OpenVMS database program development and VSI COBOL for OpenVMS database concepts.
- Chapter 6, describes tips and techniques you can use to improve program performance and reduce development and debugging time.
- Chapter 7, describes commands and generic DML statements you use to debug and test your COBOL program's DML statements.
- Chapter 8, provides examples of how you populate and manipulate a database.
- Appendix A, lists COBOL reserved words related to VSI COBOL for OpenVMS database programming, and those shared by database programs and other programs.
- Glossary of Oracle DBMS-Related Terms, provides a glossary of terms related to VSI COBOL for OpenVMS database programs.

4. Related Documents

The following documents contain additional information directly related to various topics in this manual:

- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/]

  Describes the concepts and rules of the VSI COBOL for OpenVMS programming language under the supported operating systems.

- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/]

  Describes how to develop VSI COBOL for OpenVMS programs and how to use supported operating system features from the VSI COBOL for OpenVMS language.

- *VSI COBOL Installation Guide* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-installation-guide/]

  Provides information on how to install VSI COBOL for OpenVMS on the Alpha and OpenVMS I64 operating systems.

- *COBOL for OpenVMS VAX Systems Installation Guide*

  Provides information on how to install VSI COBOL for OpenVMS on the OpenVMS VAX operating system.

- *Introduction to Oracle CODASYL DBMS*

  Introduces the features of Oracle CODASYL DBMS and provides a summary of product components. It also supplies a documentation directory, a glossary, and a master index for the entire documentation set.

- *Oracle CODASYL DBMS Database Administration Reference Manual*

  Describes the syntax and function of the data definition language (DDL) used to write schemas, storage schemas, subschemas, and security schemas. It explains the use of the DDL compiler to compile data definitions into the Oracle CDD/Repository data dictionary, the commands used in loading and unloading a database, conditional expressions, and the OpenVMS data types supported by Oracle CODASYL DBMS. Finally, the manual describes the syntax and function of each DBO command, and the DBALTER and DRU utilities.

  In addition, there is a self-paced demonstration database, which is configured to show some of the features of Oracle CODASYL DBMS. It is designed to be used with the PARTS database, which is included in the Oracle CODASYL DBMS documentation set.

- *Oracle CODASYL DBMS Programming Guide*

  Describes how to program against an Oracle CODASYL DBMS database. The manual explains how to use the Database Query utility (DBQ) to test program logic, how to embed data manipulation language statements into programs and compile them with the DML precompiler, and how to use the callable system subroutines.

- *Oracle CODASYL DBMS Programming Reference Manual*

  Describes the syntax and functions of the Oracle CODASYL DBMS data manipulation language (DML), interactive use of the Database Query utility (DBQ), and use of the callable system subroutines. The manual also describes the syntax and functions of the FORTRAN DML (FDML) statements.

- *Using Oracle CDD/Repository on OpenVMS Systems*

  A guide for repository users who store, maintain, and analyze repository information in Oracle CDD/Repository using the Common Dictionary Operator (CDO) utility.

- **The OpenVMS documentation set**

- **The Oracle CDD/Repository documentation**

5. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

6. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: `<docinfo@vmssoftware.com>`. Users who have VSI OpenVMS support contracts through VSI can contact `<support@vmssoftware.com>` for help with this product.

7. Conventions

The following product names may appear in this manual:

- OpenVMS Industry Standard 64 for Integrity servers
- OpenVMS I64
- I64

All three names — the longer form and the two abbreviated forms — refer to the version of the OpenVMS operating system that runs on the Intel ® Itanium ® architecture.

The following typographic conventions may be used in this manual:

Conventions	Meaning
< >	Angle brackets signal the end of a section of system-specific information. The beginning of a system-specific section is identified in the text or header as Alpha- or I64- or as VAX-specific.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed. Most program examples are shown in VSI terminal format, rather than in ANSI standard format.
quotation mark	The quotation mark is used to refer to the double quotation mark character (").
apostrophe	The apostrophe is used to refer to the single quotation mark character (’).
`user input`	In examples of this document, user input (what you enter) is shown as **`monospaced text`**
bold type	Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic type	Italic type indicates important information, complete titles of manuals, or variables. Variables include generic terms (lowercase variable elements in syntax) when referred to in text; and information that varies in system output (Internal error *number*), in command lines (/PRODUCER= *name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
Ctrl/*x*	The symbol **Ctrl/*x*** indicates that you hold down the key labeled CTRL while you simultaneously press another key; for example, **Ctrl/C**, **Ctrl/O**.
$	The dollar sign ($) represents the OpenVMS system prompt.

8. Acknowledgment

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

Chapter 1. Program Organization and Structure

The fundamental elements of the COBOL language are described in the current ANSI standard and the VSI COBOL for OpenVMS basic documentation set. The elements specific to database programming have been collected and concentrated in this optional book for the convenience of programmers who write COBOL database programs.

1.1. Program Structure

Figure 1.1 shows the basic structure of a COBOL program, which is organized in divisions, sections, paragraphs, sentences, and entries.

Figure 1.1. Structure of a COBOL Program

```
IDENTIFICATION DIVISION.
   PROGRAM-ID. program-name.
   AUTHOR.
   INSTALLATION.
   DATE-WRITTEN.
   DATE-COMPILED.
   SECURITY.

ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER.
   OBJECT-COMPUTER.
   SPECIAL-NAMES.
 INPUT-OUTPUT SECTION.
   FILE-CONTROL.
   I-O-CONTROL.


DATA DIVISION.
 SUBSCHEMA SECTION.
   subschema entries and keeplist entries
 FILE SECTION.
   file and record description entries
   report file description entries
   sort-merge file and record description entries
 WORKING-STORAGE SECTION.
   record description entries
 LINKAGE SECTION.
   record description entries
 REPORT SECTION.
   report and report group description entries.
 SCREEN SECTION.  (Alpha, I64)
   screen description entries  (Alpha, I64)
PROCEDURE DIVISION.
 DECLARATIVES.
   sections
     paragraphs
       sentences
```

```
END DECLARATIVES.
      .
      .
      .
  sections
    paragraphs
      sentences
      .
      .
      .
END PROGRAM header
```

The Data Division can contain a special SUB-SCHEMA SECTION section header for the Oracle CODASYL DBMS program, as shown in Figure 1.1.

A Data Division entry begins with a level indicator or level-number and is followed, in order, by:

1. A space
2. The name of a data item or file connector
3. A sequence of independent descriptive clauses
4. A separator period

The COBOL level indicators are as follows:

- DB (for subschema entries)
- LD (for keeplist entries)
- *FD* (for file description entries)
- *SD* (for sort-merge file description entries)
- *RD* (for report file description entries)

The first two level indicators in this list, DB and LD, are used in Oracle CODASYL DBMS database programs. FD, SD, and RD can be in database programs as well as in other programs.

Level indicators start in Area A.

Entries that begin with level-numbers are called data description entries. The level-number values are 01 to 49, 66, 77, and 88. Level-numbers 01 to 09 can be represented as one- or two-digit numbers.

Level 01 and 77 data description entries begin in Area A. All other data description entries can begin anywhere to the right of Margin A. Indentation has no effect on level-number magnitude; it merely enhances readability.

1.2. VSI COBOL for OpenVMS Data Manipulation Language (DML)

The VSI COBOL for OpenVMS data manipulation language (DML) is a programming language extension that provides a way for a COBOL application program to access a database. An VSI COBOL

for OpenVMS database application program contains DML statements that tell the Database Control System (DBCS) what to do with specified data; the DBCS provides all database processing control at run time. The four classes of DML statements are *data definition*, *control*, *retrieval*, and *update*. An explanation of each class follows, together with important definitions of members of that class:

- **Data definition –** These entries define the specific part of the database to be accessed by the application program and any keeplists needed to navigate through it. The entries also result in the creation of a database user work area (UWA). Transfer of data between your program and the database takes place in the UWA. Your program delivers data for the DBCS to this area; it is here that the DBCS places data requested from the database for retrieval to your program.

SUB-SCHEMA SECTION	Is the first section of the Data Division. It contains two paragraphs: the Subschema entry (DB) and the Keeplist Description entry (LD).
DB	Names the target subschema, translates subschema record descriptions to compatible VSI COBOL for OpenVMS record descriptions, and creates a user work area (UWA).
LD	Names a keeplist to help you navigate through the database.

For more information on these entries see Chapter 3.

- **Control –** The DML control functions tell the DBCS when and how to begin or end a database transaction.

COMMIT	Terminates your transaction, makes permanent all changes made to the database since the last quiet point, and establishes a new quiet point for the run unit.
READY	Prepares selected realms for use.
ROLLBACK	Ends your transaction, cancels all changes made to the database since the start of your transaction, empties all keeplists, and nulls all currency indicators.

- **Retrieval –** The DML retrieval functions are used to find a record in the database and, if necessary, retain the record in the user work area (UWA) for later use.

FIND	Locates a record in the database.
FIND ALL	Locates all specified records in the database and puts them in a keeplist.
FETCH	Locates a record in the database, retrieves its data item values, and places them in the user work area (UWA).
FREE	Releases references to records.
GET	Retrieves data item values of a previously located record and places them in the user work area (UWA).
KEEP	Remembers a record so you can later refer to it.

Records can be found in several ways in the database. By using a Record Selection Expression in a FIND or FETCH statement, a program has four formats to choose from: (1) database key identifier access, (2) set owner access, (3) record search access, or (4) DB-KEY access. Chapter 4: *Procedure Division*, explains these in detail.

A COBOL program can sequentially search the database or individual realm. In all cases, once a record is found by the COBOL application program, the DBCS sets a currency indicator to hold the database key value of that record or the position of that record. The COBOL program can indirectly use this value in KEEP, FIND ALL, or FREE statements or use the RETAINING option as a placemarker to help the program navigate through the database.

- **Update –** These functions allow the creation, modification, and deletion of database records.

CONNECT	Makes a record a member in one or more sets.
DISCONNECT	Removes a record from one or more sets.
ERASE	Deletes records from the database.
MODIFY	Changes the contents of a record in the database.
RECONNECT	Moves a record from one occurrence of a set type to another (possibly the same) occurrence.
STORE	Adds a record to the database.

Chapter 4 discusses the effects of the schema data definition language (DDL) INSERTION and RETENTION options on each of the DML update verbs.

Once a record has been located by a COBOL program, it can be changed or even erased from the database. DML programming operations also change the fundamental relationships within sets, causing records to change as well. For example, each set is owned by a record or Oracle CODASYL DBMS itself. If the program erases a record that is the owner of the set, all member records may also be deleted.

Section 4.5 contains more information on DML statements, database conditional expressions, and the special registers DB-CONDITION, DB-CURRENT-RECORD-NAME, DB-CURRENT-RECORD-ID, DB-UWA, and DB-KEY.

1.3. Creating a VSI COBOL for OpenVMS DML Program

When you create a VSI COBOL for OpenVMS DML program, you must include the SUB-SCHEMA SECTION entry as the first section in the Data Division. The SUB-SCHEMA SECTION is followed by a DB statement and any LD statements. The Procedure Division contains all occurrences of the DML verbs.

1.4. Compiling a VSI COBOL for OpenVMS DML Program

Your database administrator (DBA) creates schema and subschema definitions in Oracle CDD/Repository. These record definitions are defined in DMU format and are intended to serve all OpenVMS languages that might access them. In this format, the record definitions are not compatible with COBOL

record definitions. Therefore, when the VSI COBOL for OpenVMS compiler retrieves the subschema definition from Oracle CDD/Repository, it translates the file into an internal form acceptable to the VSI COBOL for OpenVMS compiler.

If the translation results in compiler errors, they will probably be fatal.

You should alert your DBA to any errors resulting from a DB statement.

You can define the logical name CDD$DEFAULT as the starting schema node in Oracle CDD/Repository. There is only one logical name translation in the DB statement for schema-name. If you do not define it, CDD$TOP is the default.

Note

You must recompile a VSI COBOL for OpenVMS DML program each time the subschema referenced by a DB statement is created. At compile time, the date and time of subschema creation (date and time stamps) are included with the translated subschema record definitions. If you do not recompile, your program will receive a fatal error at run time.

1.4.1. Copying Database Records in a VSI COBOL for OpenVMS Program

A separately compiled VSI COBOL for OpenVMS database program must include the SUB-SCHEMA SECTION header and only one DB statement. The compiler copies and translates the record, set, and realm definitions in the subschema named by the DB statement into compatible VSI COBOL for OpenVMS record definitions. You will not see any database record definitions listed immediately following the DB statement. The translated record, set, and realm definitions are only in the compiler's subschema map listing. To list these definitions in your program listing, use the /MAP compiler command line qualifier.

1.4.2. Using the /MAP Compiler Qualifier

Use the /MAP compiler qualifier to generate a subschema map containing a translated subschema listing. Section 7.3 contains two subschema map listings on Alpha and I64 and two on VAX and explains how to read them. The following example compiles DBPROG and creates a listing that includes a subschema map:

```
$ COBOL/LIST/MAP DBPROG
```

1.5. Linking a VSI COBOL for OpenVMS DML Program

VSI COBOL for OpenVMS DML programs must be linked with the shareable Oracle CODASYL DBMS Library (SYS$LIBRARY:DBMDML/OPT). This library was created as part of the Oracle CODASYL DBMS installation procedure. Therefore, to link a VSI COBOL for OpenVMS DML object program named DMLPROG.OBJ with the shareable Oracle CODASYL DBMS Library, you use this DCL command:

```
$ LINK DMLPROG,SYS$LIBRARY:DBMDML/OPT
```

This file name may vary depending on the version of DBMS you are using. For example, if you are using DBMS Version 7.2, the command would be the following:

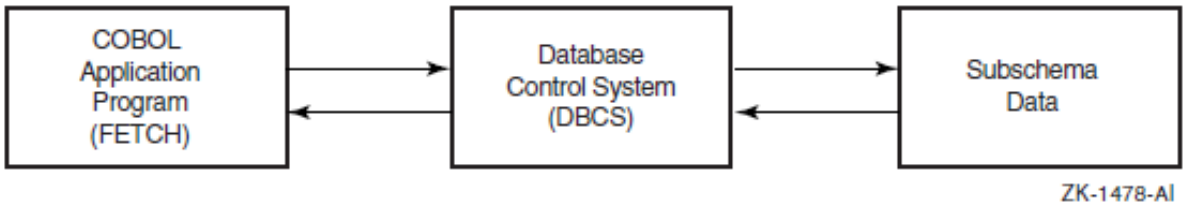
```
$ LINK DMLPROG,SYS$LIBRARY:DBMDML72/OPT
```

1.6. Running a VSI COBOL for OpenVMS DML Program

You use the DCL command RUN to execute your VSI COBOL for OpenVMS DML program. At run time, the Database Control System (DBCS) fills a variety of roles in VSI COBOL for OpenVMS. Its major functions are to monitor database usage, act as an intermediary between VSI COBOL for OpenVMS and the OpenVMS operating system, and manipulate database records on behalf of user programs. Upon execution of the first DML statement, the DBCS implicitly executes a BIND statement that links the run unit to the database. If the BIND statement is unsuccessful, a database exception occurs.

The DBCS also enforces the subschema view of the database. For example, a database schema record may contain 20 data items. However, a subschema record may only define 10 of those data items. If a FETCH statement references this record, the DBCS only retrieves those defined 10 data items and makes them available to the COBOL program in the user work area (UWA). The other 10 items are not available to the COBOL program. Figure 1.2 illustrates the run-time relationships between an application program requesting subschema data (a FETCH statement, for example), the DBCS, and the data the subschema describes.

Figure 1.2. Database and Application Program Relationship



Chapter 2. Database Programming Elements of VSI COBOL for OpenVMS

In your database programming project you will need familiarity with:

- Database-related user-defined words
- Database-related reserved words

2.1. Database-Related User-Defined Words

A user-defined word is a COBOL word that you must supply to satisfy the format of a clause or statement. This word consists of characters selected from the set A to Z, 0 to 9, the currency sign ($), underline (_), and hyphen (-). Throughout this manual, and except where specific rules apply, the hyphen (-) and the underline (_) are treated as the same character in a user-defined word. The underline (_), however, can begin or end a user-defined word, and the hyphen (-) cannot. By convention, names containing a currency sign ($) are reserved for VSI.

Within a given source program, but excluding any contained program, each database-related user-defined word belongs to one of the following disjoint sets:

Keeplist-names
Realm-names
Schema-names
Set-names
Sub-schema-names
Record-names
Data-item-names

Each user-defined database-related word in a program can belong to only one of these sets. User-defined words in each set must be unique, except as described in the rules for uniqueness of reference. (Refer to the section on Uniqueness of Reference in the Procedure Division chapter of the *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/]. The same chapter defines user-defined names).

Table 2.1 provides brief descriptions of the COBOL database-related user-defined words.

Table 2.1. COBOL Database-Related User-Defined Words

User-Defined Word Set	Purpose
Keeplist-name	Names a list of database keys used by the run unit to lock records for later use.
Realm-name	Names a database realm. (See the READY statement in the Procedure Division chapter.)
Schema-name	Names a database schema. (See the DB statement in the Data Division chapter.)
Set-name	Identifies a database set type.

User-Defined Word Set	Purpose
Sub-schema-name	Names a database subschema. (See the DB statement in the Data Division chapter.)
Record-name	Names a database record type.
Data-item-name	Names a data-item that is defined for a record type.

2.2. Database-Related Reserved Words

A reserved word can be used only as specified in the general formats. It cannot be a user-defined word. (See the appendix that lists Reserved Words.) Among the COBOL reserved words are *required* words, *optional* words, and *special-purpose* words. One of the special-purpose words, DB-CONDITION, is of special interest to the database programmer.

2.2.1. DB-CONDITION

The reserved word DB-CONDITION names a database exception condition register. It is a longword COMP item represented by PIC S9(9) USAGE IS COMP. The execution of every COBOL data manipulation language (DML) statement causes the Database Control System (DBCS) to place a status code value in this register indicating either a successful condition or an exception condition.

Before the program executes the first DML statement, the value of DB-CONDITION is initialized to DBM$_NOT_BOUND. For an explanation of this and other Oracle CODASYL DBMS status codes, refer to the Oracle CODASYL DBMS documentation set.

If a DML statement causes an exception condition, the return status code in DB-CONDITION equals the condition value. If the execution of a DML statement does not result in a database exception condition, DB-CONDITION contains a successful return status. Procedure Division statements can access the values in this register; however, only the DBCS can change the value.

2.2.2. DB-CURRENT-RECORD-NAME

This reserved word names a database register. It consists of 31 alphanumeric characters represented by PIC X(31) USAGE IS DISPLAY. The execution of COBOL data manipulation language (DML) statements that alter currency indicators causes the Database Control System (DBCS) to place a value in this register.

If the currency indicator for the run unit is not null, the register contains the name of the record type of the current record of the run unit. If the currency indicator for the run unit is null, the register contains spaces. Procedure Division statements can access the values in this register; however, only the DBCS can change the value.

2.2.3. DB-CURRENT-RECORD-ID

The reserved word DB-CURRENT-RECORD-ID names a database register. It consists of a word COMP item represented by PIC S9(4) USAGE IS COMP. The execution of COBOL data manipulation language (DML) statements that alter currency indicators causes the Database Control System (DBCS) to place a value in this register.

If the currency indicator for the run unit is not null, the register contains the subschema User ID number (UID) of the record type of the current record of the run unit. If the currency indicator for the run unit

is null, the register contains zero. Procedure Division statements can access the values in this register; however, only the DBCS can change the value.

2.2.4. DB-KEY

The reserved word DB-KEY names a database register. It consists of a quadword COMP item represented by PIC S9(18) USAGE IS COMP. The execution of the COBOL data manipulation language (DML) statements FETCH, FIND, and STORE causes the Database Control System (DBCS) to place a value in this register.

The DB-KEY special register contains three values. To access the individual values, move the DB-KEY register to a record as follows:

```
01  DATABASE-KEY
    02  LINE-NUMBER     PIC 9(4) USAGE IS COMP.
    02  PAGE-NUMBER     PIC 9(9) USAGE IS COMP.
    02  AREA-NUMBER     PIC 9(4) USAGE IS COMP.
```

2.2.5. DB-UWA

The reserved word DB-UWA names a database register. It consists of 108 alphanumeric characters represented by PIC X(108) USAGE IS DISPLAY. The Database Control System (DBCS) makes data items available to your program through the DB-UWA record delivery area. The DB-UWA register can be used with callable DBQ routines. Procedure Division statements can access the values in this register; however, only the DBCS can change the value. For more information, refer to the discussion of the DBO/WORK_AREA command in the Oracle CODASYL DBMS documentation set.

Chapter 3. Data Division

The logical and physical concepts that apply to the Data Division in any COBOL program also apply to your database programs. The *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/] presents the general formats for Data Division entries and clauses, describes their basic elements, and lists rules of use. Information of special interest to the database programmer will be found here.

The Data Division defines the data processed by your COBOL program in both physical and logical terms. It also specifies whether the data is contained in files, a database, or Oracle CDD/Repository, or is developed only for local use in your program.

The Subschema Section specifies data contained in a database or Oracle CDD/Repository. The Working-Storage and Linkage Sections contain data description entries, which describe characteristics of data developed for use in your program.

3.1. DATA DIVISION General Format and Rules

The last five COBOL sections are described and discussed in the *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/] chapter on Data Division.

Function

The Data Division describes data the program creates, receives as input, manipulates, and produces as output.

General Format

DATA DIVISION.

[SUB-SCHEMA SECTION. | sub-schema-entry. | [{ keeplist-entry }...]]

[FILE SECTION. | [file-description-entry{ record-description-entry }...]... | [report-file-description-entry]... | [sort-merge-file-description-entry{ record-description-entry }...]...]

[WORKING-STORAGE SECTION. [record-description-entry]...]

[LINKAGE SECTION. [record-description-entry]...]

[REPORT SECTION. [report-description-entry{ report-group-description-entry }...]]

[SCREEN SECTION. [screen-description-entry]... (Alpha, I64)]

Syntax Rules

1. The Data Division follows the Environment Division.
2. The reserved words DATA DIVISION, followed by a separator period, identify and begin the Data Division.

General Rules

1. The Data Division has six sections. These sections must be in the following order:

SUB-SCHEMA SECTION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
REPORT SECTION.
SCREEN SECTION. (Alpha, I64)

Subschema Section

1. The Subschema Section names the subschema you want to use. It describes a logical view of the database as it is to be accessed by the COBOL program. It begins with the Subschema Section header containing the reserved words SUB-SCHEMA SECTION followed by a period (.) separator character.

2. The Subschema entry follows the Subschema Section header.

3. The Subschema entry consists of a level indicator (DB), a sub-schema-name, the reserved word WITHIN, a schema-name, and an optional database clause.

4. A DB entry specifies the following:

   - Which subschema contains the descriptions of the data to be made available to the COBOL program

   - Which schema contains the subschema definition in Oracle CDD/Repository

   - The name of the database that the program is to be run against

   - The stream name

5. A run unit can declare only one Subschema (DB) entry.

6. Keeplist entries follow the Subschema (DB) entry. A keeplist is a table containing an ordered list of database key values.

7. Each database key value represents a record occurrence in the Oracle CODASYL DBMS database. A keeplist can contain the same database key value any number of times. Any number of keeplists can contain the same database key value any number of times.

8. A database key is added to the end of a keeplist by the KEEP or the FIND ALL statement and selectively removed by the FREE statement. One FIND ALL statement can add multiple database keys to a keeplist.

9. The number of entries in a keeplist is called the **cardinality** of the keeplist. A keeplist with zero entries is called an **empty keeplist**. The order of the entries in the keeplist reflects their sequence of insertion. As entries are added, the cardinality of the keeplist increases. As entries are removed, the cardinality decreases.

10. A COBOL program can reference a keeplist by using a database key identifier.

11. The Database Control System (DBCS) creates and maintains all keeplists.

12. No other run unit can update a record whose database key value is in a keeplist. The DBCS locks each record that is in a keeplist. The record remains locked until the program frees it or until the

program executes a COMMIT (without the RETAINING option) or ROLLBACK statement, or until the program terminates.

Additional References

Refer to the *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/] for:

- REPORT clause
- VALUE IS clause
- CALL statement
- User-defined words

DB (Subschema Description)

DB (Subschema Description) — The Subschema entry allows a program to access a subschema in Oracle CDD/Repository under the schema name.

General Format

DB sub-schema-name WITHIN schema-name

[FOR database-name] [{ THRU | THROUGH } `stream-name`].

sub-schema-name
references a subschema name in Oracle CDD/Repository under the schema name. See Technical Notes for more information.

schema-name
references a schema name in Oracle CDD/Repository. See Technical Notes for more information.

database-name
represents a complete or partial file specification defining the database occurrence. A database occurrence is a root file. At run time, database-name points the Database Control System (DBCS) to the root file. See Technical Notes for more information.

stream-name
represents the name of the stream. See Technical Notes for more information.

General Rules

1. The compiler finds the *schema-name* within Oracle CDD/Repository under your CDD$DEFAULT and includes these *sub-schema-name* definitions in your program: realm names, set names, record names, database data-names. These definitions are user-defined words. Other definitions include the database registers DB-CONDITION, DB-CURRENT-RECORD-NAME, DB-UWA, DB-CURRENT-RECORD-ID, and DB-KEY. Subschema definitions are also called *database objects.*
2. Database objects implicitly have the global attribute. Therefore, any program contained within a program declaring the DB statement can access any of the declaring program's database objects.
3. Only one DB statement is allowed for each separately compiled program; however, more than one DB statement is allowed in the run unit if its separately compiled programs define identical DB statements, or if each separately compiled program uses a stream.

4. Database objects either explicitly or implicitly defined in the subschema section are external objects by default. However, these objects can only be referenced by a program defining the DB statement or a program contained within that program.

5. When you compile a program containing a DB statement, the resulting .OBJ file includes the subschema date and time stamp assigned to the *sub-schema-name* when it was inserted in Oracle CDD/Repository. At run time, the DBCS finds the root file and binds the program to the subschema identified by *sub-schema-name*. If the program does not contain the same date and time as the subschema identified by *sub-schema-name*, then compile-time subschema definitions may not be the same as the run-time definitions; therefore, an exception condition occurs. This security check prevents your program from accessing the database with obsolete subschema definitions.

6. If you include the FOR database-name clause, the DBCS determines if *database-name* is a logical name or a file specification. If it is a partial file specification, the default file type is .ROO.

7. If you omit the FOR database-name clause, the DBCS creates a *database-name* from *schema-name*. The DBCS determines if *schema-name* is a logical name. If it is, a translation is done to determine the *database-name*. If *schema-name* is not a logical name, the DBCS creates a *database-name* by appending .ROO to the first nine characters of *schema-name*.

8. If a separately compiled program in the run unit uses a stream, then all separately compiled programs in the same run unit must use streams. Otherwise, a run-time error occurs.

Technical Notes

- Table 3.1 describes the representation of data types in the Oracle CODASYL DBMS data definition language (DDL) compiler and the VSI COBOL for OpenVMS for OpenVMS Alpha, I64, or VAX compiler. Additionally, this table lists all data types that can be specified using the DDL data definition language with the corresponding COBOL data item picture. Note that COBOL does not have an equivalent specification for some data types.

Table 3.1. VSI COBOL for OpenVMS and DDL Utility Data Type Equivalences

DDL	COBOL
SIGNED BYTE	(a)
UNSIGNED BYTE	(a)
SIGNED WORD	S9(4) COMP
UNSIGNED WORD	(b)
SIGNED LONGWORD	S9(9) COMP
UNSIGNED LONGWORD	(b)
SIGNED QUADWORD	S9(18) COMP

Legend:

l – The total number of digits for the item.
s – The decimal offset to 1.
(a) – COBOL has no equivalent for this data type. A fatal diagnostic is issued for such an item, which is part of a subschema record description. The compiler treats this item as if it had been specified as an alphanumeric data item that occupies the same number of bytes.
(b) – COBOL has no exact equivalent for this data type. A warning diagnostic is issued for such an item, which is part of a subschema record description. The compiler treats this item as if it had been specified as the equivalent unsigned COMP item.

DDL	COBOL
UNSIGNED QUADWORD	(b)
SIGNED OCTAWORD	(a)
UNSIGNED OCTAWORD	(a)
FLOATING	COMP-1
FLOATING COMPLEX	(a)
D_FLOATING	COMP-2 (/FLOAT=D on Alpha, I64, VAX)
D_FLOATING COMPLEX	(a)
G_FLOATING	COMP-2 (/FLOAT=G on Alpha, I64)
G_FLOATING COMPLEX	(a)
H_FLOATING	(a)
H_FLOATING COMPLEX	(a)
CHARACTER l	X(l)
UNSIGNED NUMERIC l s	9(m)V9(n)
LEFT SEPARATE NUMERIC l s	S9(m)V9(n) LEADING SEPARATE
LEFT OVERPUNCHED NUMERIC l s	S9(m)V9(n) LEADING
RIGHT SEPARATE NUMERIC l s	S9(m)V9(n) TRAILING SEPARATE
RIGHT OVERPUNCHED NUMERIC l s	S9(m)V9(n) TRAILING
ZONED NUMERIC l s	(a)
PACKED DECIMAL l s	S9(m)V9(n) COMP-3
Legend: **l – The total number of digits for the item.** **s – The decimal offset to 1.** **(a) – COBOL has no equivalent for this data type. A fatal diagnostic is issued for such an item, which is part of a subschema record description. The compiler treats this item as if it had been specified as an alphanumeric data item that occupies the same number of bytes.** **(b) – COBOL has no exact equivalent for this data type. A warning diagnostic is issued for such an item, which is part of a subschema record description. The compiler treats this item as if it had been specified as the equivalent unsigned COMP item.**	

The method for describing the assumed decimal point is different in the two products. In a COBOL picture, the decimal position is directly indicated by the symbol V or implied by the symbol P. In DDL, scaled numbers are specified by two integers. The first integer represents the total number of decimal digits that the item represents. The second integer represents the decimal offset to the first integer.

For example, the COBOL data item described by PIC 9(4)V99 is equivalent to the DDL entry TYPE IS UNSIGNED NUMERIC 6 -2. Similarly, the DDL entry TYPE IS LEFT SEPARATE NUMERIC 6 2 is equivalent to the COBOL description PIC S9(6)PP SIGN IS LEADING SEPARATE. The items described using the CDD INDEXED FOR COBOL BY clause become COBOL index-names.

- *Schema-name*, *sub-schema-name*, *database-name*, and *stream-name* can be a nonnumeric literal or a COBOL word formed according to the rules for user-defined names. *Database-name* represents a complete or partial file specification. If any of these names is not a literal, the compiler:

  - Translates hyphens in the COBOL word to underline characters
  - Translates lowercase alphabetic characters to uppercase
  - Treats the word as if it were enclosed in quotation marks

- Most of the information described in the remaining Technical Notes is included in the subschema map. See Chapter 1, for information on the /MAP qualifier.
- You can use this Database Operator (DBO) utility command to display date and time stamp information on *schema-name* and *sub-schema-name*:

  ```
  $ DBO/REPORT
  ```

  Additional information is available with the HELP DBO/REPORT command.

- You can use the DBO utility command to create a *sub-schema-name* output file from Oracle CDD/Repository:

  ```
  $ DBO/EXTRACT schema-name/SUB-SCHEMAS=sub-schema-name/OUTPUT=filename -
  $_ /OPTION=FULL
  $ PRINT filename
  ```

  Additional information is available with the HELP DBO/EXTRACT command.

- You can use the DBO utility command to create a *sub-schema-name* output file from the root file:

  ```
  $ DBO/DUMP database-name/SUB-SCHEMAS=sub-schema-name/OUTPUT=filename
  $ PRINT filename
  ```

  Additional information is available with the HELP DBO/DUMP command.

Additional Reference

Refer to the Oracle CODASYL DBMS documentation set for more information.

Examples

1. This example references the default subschema in the PARTS database:

   ```
   DB DEFAULT_SUB-SCHEMA WITHIN PARTS.
   ```

2. This example references the NEW database that contains the PARTS schema and the PARTSS1 subschema:

   ```
   DB PARTSS1 WITHIN PARTS FOR "DB1:[COBOL88]NEW.ROO".
   ```

3. This example references the default subschema in the PARTS database through the stream STREAM1.

   ```
   DB DEFAULT_SUB-SCHEMA WITHIN PARTS THRU "STREAM1".
   ```

LD (Keeplist Description)

LD (Keeplist Description) — The Keeplist Description entry names a keeplist.

General Format

LD keeplist-name [LIMIT IS integer].

keeplist-name
is a user-defined name.

integer
is a positive integer.

Syntax Rules

1. The LD entry can appear only in the Subschema Section following the DB entry or another LD entry.

2. The LIMIT clause is for documentation only. The compiler ignores this clause.

General Rules

1. *Keeplist-name* implicitly has the global attribute. Any program defining the name, and any program contained within that defining program, can reference *keeplist-name*.

2. A keeplist is a table containing an ordered list of database key values.

3. At program initiation, each existing keeplist is empty.

4. A KEEP or a FIND ALL statement adds a database key value to the end of *keeplist-name*. FIND ALL can add multiple database keys to a keeplist.

5. A FREE statement selectively removes a database key value from *keeplist-name*.

6. A keeplist with zero entries is called an **empty keeplist.** The order of the entries in the keeplist reflects their sequence of insertion. As entries are added, the cardinality increases. As entries are removed, the cardinality decreases.

7. A COBOL program can reference a keeplist entry by using a database key identifier.

8. The Database Control System (DBCS) creates and maintains all keeplists.

9. Each value in *keeplist-name* represents a database record in the database.

10. A keeplist can contain the same database key value more than once.

11. More than one keeplist can contain the same database key value at the same time.

12. No other run unit can update a record whose database key value is in a keeplist. Every entry in the keeplist causes the DBCS to perform a retrieval lock on the record it identifies. The record remains locked until the entry is freed, or until the program executes a COMMIT (without the RETAINING option) or ROLLBACK statement, or until the program terminates. If another run unit references a locked record, the DBCS forces the run unit to wait until the record is unlocked.

Additional Reference

Section 4.4, on database key identifiers.

Example

The following example defines three keeplists to navigate through the PARTSS1 subschema: KEEP-COMPONENT, K-EMPLOYEE, and KL-ID:

```
DB PARTSS1 WITHIN PARTS.
LD KEEP-COMPONENT.
LD K-EMPLOYEE.
LD KL-ID.
```

Chapter 4. Procedure Division

The Procedure Division of your COBOL database programs may contain declarative and nondeclarative procedures. These may include:

- Directive and imperative statements
- Conditional statements
- Database key identifiers
- Explicit and implicit scope terminators
- Section 4.6, on record selection expressions.

4.1. COBOL Statements for the Database Programmer

There are four types of COBOL statements:

- *Compiler-directing* statements specify an action taken by the compiler during compilation.
- *Imperative* statements specify an unconditional action taken by the object program at run time.
- *Conditional* statements specify a conditional action taken by the object program at run time; the action depends upon a truth value that is generated by the program. (A truth value is either a yes or no answer to the question, “Is the condition true?”)
- *Delimited-scope*statements specify their explicit scope terminator.

Table 4.1 shows the three types of COBOL statements (conditional, imperative, delimited-scope) that are considered particularly relevant to database programming.

Table 4.1. Types of COBOL Statements

Type	Verb
Conditional	COMMIT ([NOT] ON ERROR) CONNECT ([NOT] ON ERROR) DISCONNECT ([NOT] ON ERROR) ERASE ([NOT] ON ERROR) FETCH ([NOT] AT END or [NOT] ON ERROR) FIND ([NOT] AT END or [NOT] ON ERROR) FREE ([NOT] ON ERROR) GET ([NOT] ON ERROR) KEEP ([NOT] ON ERROR) MODIFY ([NOT] ON ERROR) READY ([NOT] ON ERROR) RECONNECT ([NOT] ON ERROR)
Legend: **(1) Without the optional [NOT] ON ERROR phrase** **(2) Without the optional [NOT] AT END or [NOT] ON ERROR phrase**	

Type	Verb
	ROLLBACK ([NOT] ON ERROR) STORE ([NOT] ON ERROR)
Imperative	COMMIT (1) CONNECT (1) DISCONNECT (1) ERASE (1) FETCH (2) FIND (2) FREE (1) GET (1) KEEP (1) MODIFY (1) READY (1) RECONNECT (1) ROLLBACK (1) STORE (1)
Delimited-Scope	COMMIT (END-COMMIT) CONNECT (END-CONNECT) DISCONNECT (END-DISCONNECT) ERASE (END-ERASE) FETCH (END-FETCH) FIND (END-FIND) FREE (END-FREE) GET (END-GET) KEEP (END-KEEP) MODIFY (END-MODIFY) READY (END-READY) RECONNECT (END-RECONNECT) ROLLBACK (END-ROLLBACK) STORE (END-STORE)
Legend: **(1) Without the optional [NOT] ON ERROR phrase** **(2) Without the optional [NOT] AT END or [NOT] ON ERROR phrase**	

Like statements, COBOL sentences also can be compiler-directing, imperative, or conditional. Sentence type depends upon the types of clauses the statement contains. Table 4.2 summarizes the contents of the three types of COBOL sentences. The remaining text in this section discusses each type of statement and sentence in greater detail.

Table 4.2. Contents of COBOL Sentences

Type	Contents of Sentence
Imperative	One or more consecutive imperative statements ending with a period
Conditional	One or more conditional statements, optionally preceded by an imperative statement, terminated by the separator period
Compiler-Directing	Only one compiler-directing statement ending with a period

4.1.1. Compiler-Directing Statements and Sentences

A compiler-directing statement causes the compiler to take an action during compilation. The verbs COPY, REPLACE, RECORD, or USE define a compiler-directing statement. When it is part of a sentence that contains more than one statement, the COPY, REPLACE, RECORD, or USE statement must be the last statement in the sentence.

A compiler-directing sentence is one COPY, REPLACE, RECORD, or USE statement that ends with a period.

4.1.2. Imperative and Conditional Statements and Sentences

An imperative statement specifies an unconditional action for the program. It must contain a verb and the verb's operands, and cannot contain any conditional phrases. For example, the following statement is imperative:

```
READY UPDATE.
```

However, the following statement is not imperative because it contains the phrase ON ERROR, which makes the program's action conditional:

```
STORE PART-REC ON ERROR PERFORM BAD-STORE.
```

A delimited-scope statement is a special category of imperative statement used in structured programming. A delimited-scope statement is any statement that includes its explicit scope terminator. For more information, see the section on Explicit and Implicit Attributes.

In the Procedure Division rules, an imperative statement can be a sequence of consecutive imperative statements. The sequence must end with: (1) a separator period or (2) any phrase associated with a statement that contains the imperative statement. For example, the following sentence contains a sequence of two imperative statements following the AT END phrase.

```
FIND NEXT PART-REC AT END PERFORM NO-MORE-RECS
                    DISPLAY "No more records."     END-FIND.
```

An imperative sentence contains only imperative statements and ends with a separator period.

4.2. Explicit and Implicit Scope Terminators

Scope terminators delimit the scope of some Procedure Division statements.

Explicit scope terminators for database programs are as follows:

END-COMMIT	END-FIND	END-READY
END-CONNECT	END-FREE	END-RECONNECT
END-DISCONNECT	END-GET	END-ROLLBACK
END-ERASE	END-KEEP	END-STORE
END-FETCH	END-MODIFY	

Implicit scope terminators are as follows:

- At the end of a sentence: the separator period. It terminates the scope of all previously unterminated statements.

- In a statement containing another statement: the next phrase of the containing statement after the contained statement terminates the scope of all unterminated contained statements. Examples are ELSE and WHEN.

4.3. Scope of Names

A contained COBOL program can refer to a user-defined word in its containing program if the user-defined word has the global attribute. (See the section on User-Defined Words.) Some user-defined words always have the global attribute, some never have the attribute (that is, they are local), and some may or may not, depending on the use of the GLOBAL clause. The following rules explain how to use different kinds of user-defined words and what kinds of local and global *name scoping* to expect.

User-defined words in the Subschema Section are always global. The program defining this section and in any program it contains can reference these user-defined words:

- Data-name
- Keeplist-name
- Realm-name
- Record-name
- Set-name

4.4. Database Key Identifiers

Database key identifiers are indicated by the reserved words CURRENT, OFFSET, FIRST, and LAST as shown in the two formats below. FETCH and FIND may use database key identifiers to access database data records. Conditional clauses may also use them.

General Formats

Format 1—Currency Indicator Access

CURRENT [WITHIN{ record-name | set-name | realm-name }]

Format 2—Keeplist Access

{ OFFSET integer-exp | FIRST | LAST } **WITHIN keeplist-name**

General Rules

Format 1

1. *Record-name* references a database record in the subschema.
2. *Set-name* references a database set in the subschema.
3. *Realm-name* references a subschema realm.

4. Use this format to select a record whose database key value is in a currency indicator.
5. The referenced currency indicator is the currency indicator for the record, set, or realm you specify in *record-name*, *set-name*, or *realm-name*.
6. If you do not specify the WITHIN phrase, the referenced currency indicator is the currency indicator of the current record of the run unit.
7. An exception condition occurs if the referenced currency indicator is null:
   a. Run-unit currency indicator (DB-CONDITION is set to DBM$_CRUN_NULL).
   b. Record type currency indicator (DB-CONDITION is set to DBM$_CRTYP_NULL).
   c. Set type currency indicator (DB-CONDITION is set to DBM$_CSTYP_NULL).
   d. Realm currency indicator (DB-CONDITION is set to DBM$_CRELM_NULL).

   See the USE statement for information on USE FOR DB-EXCEPTION.
8. Other exception conditions occur if the referenced currency indicator points to a vacant position in the database collection:
   a. Run-unit currency indicator (DB-CONDITION is set to DBM$_CRUN_POS).
   b. Set type currency indicator (DB-CONDITION is set to DBM$_CSTYP_POS).
   c. Realm currency indicator (DB-CONDITION is set to DBM$_CRELM_POS).

   See the USE statement for information on USE FOR DB-EXCEPTION.

Format 2

1. *Integer-exp* is an arithmetic expression or integer. It refers to a position in *keeplist-name*. *Integer-exp* cannot be zero.
2. *Keeplist-name* is a keeplist defined in the Subschema Section.
3. Use this format to select a record whose database key value is in a keeplist. Because a keeplist can contain more than one entry, you must specify which keeplist entry you want.
4. *Integer-exp* can be either an integer or an arithmetic expression. Both result in a longword integer value.
5. Using the FIRST clause is equivalent to OFFSET plus 1. It references the first database key value in the keeplist.
6. Using the LAST clause is equivalent to OFFSET minus 1. It references the last database key value in the keeplist.
7. The value of *integer-exp* points to an entry in *keeplist-name*. The entry whose ordinal position in the keeplist is equal to the value of *integer-exp* is called the *referenced keeplist entry*.
   a. If *integer-exp* is positive, the ordinal position is relative to the first entry in the keeplist.
   b. If *integer-exp* is negative, the ordinal position is relative to the last entry in the keeplist.

8. An exception condition occurs if:

   a. *Integer-exp* is zero. DB-CONDITION is set to DBM$_BADZERO.

   b. The absolute value of *integer-exp* is greater than the number of database key values in the keeplist. DB-CONDITION is set to DBM$_END.

   c. The identified keeplist is empty. DB-CONDITION is set to DBM$_END.

   See the USE statement for information on the USE FOR DB-EXCEPTION statement.

4.5. Database Conditions

Database conditions allow alternate paths of control depending on the truth value of a test involving conditions specific to the database environment. The database conditions are the tenancy, member, and database key conditions.

Database exception conditions can occur during evaluation of these conditions. If a database exception condition occurs during the execution of a database condition, the Database Control System (DBCS) places a database exception condition code in the special register DB-CONDITION. This code identifies the condition. The DBCS also does the following:

1. Places the record name of database-record in the special register DB-CURRENT-RECORD-NAME.

2. Places the UID (User ID number) of the database record in DB-CURRENT-RECORD-ID.

3. Invokes an applicable USE FOR DB-EXCEPTION Declaratives procedure, if any; otherwise, the DBCS abnormally terminates the run unit (see the USE statement). Under these circumstances, the result of the test is false.

4.5.1. Tenancy Conditions

These conditions determine whether a record in the database is an owner, or member, or a tenant in one or more sets.

General Format

[NOT] [set-name] { OWNER | MEMBER | TENANT }

`set-name`
is a subschema set name.

The result of the test is true if the current record of the run unit is as follows:

- Owner of an occurrence of *set-name* (OWNER clause)

- Member of an occurrence of *set-name* (MEMBER clause)

- Owner or member of an occurrence of *set-name* (TENANT clause)

Otherwise, the result of the test is false. For example:

```
IF PART_USES OWNER PERFORM 100-PART-OWNER.
IF PART_USED_ON MEMBER ...
```

```
IF RESPONSIBLE_FOR TENANT ...
```

If NOT is used, the result of the test is reversed.

Omitting *set-name* allows all subschema set types in which the record participates to be considered in determining the truth value of the condition.

If the run-unit currency indicator is null, a database exception condition occurs and DB-CONDITION is set to DBM$_CRUN_NULL. If the run-unit currency indicator only specifies a position, a database exception condition occurs and DB-CONDITION is set to DBM$_CRUN_POS. See Section 4.8.3: Exception Conditions and the USE Statement for information on USE FOR DB-EXCEPTION.

4.5.2. Empty Condition

The empty condition determines whether member records are present in one or more sets. Only member record types defined in the Subschema Section are considered in determining the truth value of the condition.

General Format

[set-name IS] [NOT] **EMPTY**

set-name
is a subschema set name.

If *set-name* is specified, the object set is the current set of that set type. If the object set has no member records, the result of the test is true. If the object set has member records, the result of the test is false.

If *set-name* is not specified, the object sets, if any, are owned by the current record of the run unit whose set type is defined in the Subschema Section. If each object set has no member records, the result of the test is true. If any object set has member records, the result of the test is false. For example:

```
IF PART_USES EMPTY ...
IF EMPTY ...
```

If NOT is used, the result of the test is reversed.

A database exception condition occurs if either:

- The currency indicator for the run unit is null (DB-CONDITION is set to DBM$_CRUN_NULL).
- The currency indicator for the run unit specifies a position (DB-CONDITION is set to DBM $_CRUN_POS).
- The current record of run unit is not the owner of *set-name* (DB-CONDITION is set to DBM $_NOTOTYP).

See Section 4.8.3: Exception Conditions and the USE Statement for information on USE FOR DB-EXCEPTION.

4.5.3. Database Key Condition

The database key condition determines whether: (1) two database key values identify the same database record, (2) a database key value is null, or (3) a database key value is identical to any database key value in a keeplist.

General Format

`database-key IS` [NOT] { ALSO database-key | NULL | WITHIN keeplist-name }

`database-key`
references a currency indicator (see section on Database Key Identifiers) or a keeplist entry in the Subschema Section.

The result of the test is true if:

- When using the ALSO phrase, both *database-key*s reference identical database key values. This example compares a set currency indicator for ALL_PARTS to the currency indicator of the current record.

  ```
  IF CURRENT WITHIN ALL_PARTS IS ALSO CURRENT ...
  ```

- When using the NULL phrase, the currency indicator referenced by *database-key* is null. This example checks the currency indicator for the current record before storing its key in a keeplist:

  ```
  IF CURRENT IS NULL NEXT SENTENCE   ELSE    KEEP USING KEEP-LIST-A.
  ```

- When using the WITHIN phrase, any database key value in *keeplist-name* is identical to *database-key*. For example:

  ```
  IF CURRENT IS WITHIN KEEP-LIST-A            PERFORM 200-ITS-IN-THE-LIST.
  ```

Otherwise, the result of the test is false.

If NOT is used, the result of the test is reversed.

A database exception condition occurs if:

- The program identifies an invalid database key identifier. (See the Database Key Identifiers section for more information.)
- For the NULL condition, the referenced currency indicator points to a vacant position in the database or collection:
  - Realm currency indicator (DB-CONDITION is set to DBM$_CRELM_POS).
  - Set currency indicator (DB-CONDITION is set to DBM$_CSTYP_POS).
  - Run-unit currency indicator (DB-CONDITION is set to DBM$_CRUN_POS).

  See the USE statement for more information on USE FOR DB-EXCEPTION.

4.5.4. Condition Evaluation Rules

Parentheses can specify the evaluation order in complex conditions. Conditions in parentheses are evaluated first. In nested parentheses, evaluation starts with the innermost set of parentheses. It proceeds to the outermost set.

Conditions are evaluated in a hierarchical order when there are no parentheses in a complex condition. This same order applies when all sets of parentheses are at the same level (none are nested). The hierarchy is shown in the following list:

1. Values for arithmetic expressions
2. Truth values for simple conditions, in this order:
   a. Relation
   b. Class
   c. Condition-name
   d. Switch-status
   e. Sign
   f. Database
   g. Success/failure
3. Truth values for negated simple conditions
4. Truth values for combined conditions, in this order:
   a. AND logical operators
   b. OR logical operators
5. Truth values for negated combined conditions

In the absence of parentheses, the order of evaluation of consecutive operations at the same hierarchical level is from left to right.

4.6. Record Selection Expressions (RSE)

A record selection expression is used to select a record in the database. It can be used in a FETCH or FIND statement. The record thus selected becomes the current record of the run unit upon which subsequent statements may operate when accessing the database.

Refer also to the *Oracle CODASYL DBMS Programming Guide* and the *Oracle CODASYL DBMS Programming Reference Manual* for additional information.

General Formats

Format 1—Database Key Identifier Access

`database-key-identifier`

This format selects a record by a database key value held by the Database Control System (DBCS) in a currency indicator or a keeplist entry.

`database-key-identifier`
identifies a record according to the rules of database key identifiers. (See Section 4.4: Database Key Identifiers for more information.)

For example:

```
FIND CURRENT WITHIN PART-REC
```

Format 2—Set Owner Access

This format selects the record that owns a set.

OWNER WITHIN set-name

set-name
is a subschema set name. The DBCS uses the currency indicator for *set-name* to choose the owner record of that set occurrence. A database exception condition occurs if *set-name* is a singular set (DB-CONDITION is set to DBM$_SINGTYP) or if the currency indicator for the set type is null (DB-CONDITION is set to DBM$_CSTYP_NULL).

Format 3—DB Key Access

This format selects the record that is referred to by the database key contained in the special register DB-KEY.

[DBKEY]

Format 4—Record Position Access

This format selects a record by its position within a collection of records and optionally by its record type and contents.

{ FIRST | [LAST] | NEXT | PRIOR | ANY | DUPLICATE | [RELATIVE] `int-exp`}

[record-name] [WITHIN { realm-name | set-name }] [USING [rec-key]... | WHERE [bool-expression]]

bool-expression:

{ bool-alternate [OR bool-alternate]... }

bool-alternate:

{ simple-bool-relation [AND simple-bool-relation]... }

simple-bool-relation:

{ bool-condition | NOT bool-expression }

bool-expression:

{{ id | lit } { IS{ GREATER THAN OR EQUAL TO | >= | LESS THAN OR EQUAL TO | <= } | IS [NOT] { EQUAL TO | = | GREATER THAN | > | LESS THAN | < } | DOES [NOT] { CONTAIN | CONTAINS | MATCH | MATCHES } } { id | lit }}

int-exp

is an integer or arithmetic expression resulting in a longword integer value. It cannot be zero. It may be an embedded literal or an integer data-name.

realm-name

is a subschema realm name.

`record-name`

is a subschema record name.

`set-name`

is a subschema set name.

`rec-key`

is a key data item within the subschema record occurrence. The same *rec-key* can appear only once in a given USING phrase.

`bool-expression`

is a conditional expression that involves data items of the object record. It is used to specify additional requirements of a qualifying record.

`bool-alternate`

is one or more sub-expressions (simple-bool-relation). Pairs of sub-expressions are joined by the logical operator AND.

`simple-bool-relation`

is a simple-condition (bool-condit), an expression, or the negation of either.

`bool-condition`

is a relation involving two operands joined by a relational operator.

Relational operators can be one of the following:

[NOT] EQUAL (=) TO
[NOT] LESS (<) THAN
[NOT] GREATER (>) THAN
GREATER THAN OR EQUAL (>=) TO
LESS THAN OR EQUAL (<=) TO
[NOT] CONTAIN
[NOT] CONTAINS
[NOT] MATCH
[NOT] MATCHES

The relational operator CONTAINS is used to check that a given data item in the record contains the specified string anywhere within it. At least one of the operands must be the identifier of a nondatabase item or a nonnumeric literal; the other operand must be an elementary item in the record being found or fetched.

The relational operator MATCHES checks that a given item in the records matches the specified pattern string. At least one of the operands must be the identifier of a nondatabase item or a nonnumeric literal; the other must be an elementary item in the record being found or fetched.

The pattern string is formed from any character string; however, two characters, the percent sign (%) and the asterisk (*), have special meanings. If the percent sign (%) occurs in the pattern string, it will match the *current single* character in the item being matched. If the asterisk (*) occurs in the pattern string,

it will match *any number* of characters starting at the current character in the item being matched. For example:

```
FIND ALL KEEP-1 WHERE PARTDESC MATCHES '*DISK*'
FIND ALL KEEP-1 WHERE PARTDESC MATCHES '*F%%'
```

To match either the percent sign (%) or asterisk (*) in the pattern string, precede it with the caret (that is, to match percent sign (%) use caret and percent (^%), and to match asterisk (*) use caret and asterisk (^*)).

Collection Clause

The reserved word WITHIN with *set-name* or *realm-name* is called the collection clause. Use the collection clause to restrict a search to a specific collection of records in the database. If you do not use the collection clause, the DBCS searches through all the records in the database to which you have access.

The collection clause specifies the object collection. The following rules govern its use:

1. When you use WITHIN *realm-name*, realm records make up the object collection. The realm currency indicator is called the object currency indicator. The ordering of the object collection is unspecified.

2. When you use WITHIN *set-name*, the member records of the current set of the specified set type make up the object collection. The set type's currency indicator is called the object currency indicator. The ordering of the object collection is the ordering of the records in the current set of the specified set type.

3. If you use neither WITHIN *realm-name* nor WITHIN *set-name*, all records defined in the subschema make up the object collection. The object currency indicator defaults to the currency indicator for the run unit. The ordering of the object collection is unspecified.

Qualification Clause

Use *record-name* to restrict the search to records of a particular type. If you do not use *record-name*, the DBCS searches each record type in the subschema. You can further restrict the search to records with specific data item values. You do this by specifying the qualification clause. The qualification clause is either the reserved word USING followed by one or more *rec-keys*, or the reserved word WHERE followed by a Boolean expression. When USING is specified, the DBCS searches for only those database records whose key data items equal the corresponding data items in your user work area. When WHERE is specified, the DBCS searches for database records of the object record type whose item values cause the Boolean expression to evaluate to true.

The qualification clause optionally specifies the following additional rules for determining the object record type, the qualifying record, and the key data item:

1. If *record-name* is used, its record type is called the object record type.

2. *Rec-key* must be a data item defined in the subschema database record type, *record-name*.

3. If the key data item consists of one or more *rec-keys*, it describes additional criteria for selecting a record.

4. If the USING clause is used, a qualifying record is an occurrence of the object record whose corresponding variables in the user work area (UWA) equal the contents of the key data item.

5. If the WHERE clause is used, each occurrence of the object record type for which the *bool-expression* is true is called a qualifying record.

6. If you do not use the qualification clause, each occurrence of *record-name*, regardless of its contents, is called the qualifying record.

7. At least one operand in every relation condition of a WHERE Boolean expression must be an elementary item in the record being found or fetched.

Position Clause

The reserved words FIRST, LAST, NEXT, PRIOR, ANY, DUPLICATE, and RELATIVE make up the position clause. The position clause selects a specific qualifying record in the object collection. The position clause rules follow:

1. *Int-exp* refers to an embedded integer literal or to an integer variable. *Int-exp* can be either an integer or an arithmetic expression. Both result in a longword integer value. *Int-exp* cannot be zero.

2. ANY is equivalent to FIRST.

3. DUPLICATE is equivalent to NEXT.

4. Using the FIRST clause is equivalent to assigning +1 to *int-exp*. It refers to the first record in the object collection.

5. Using the LAST clause is equivalent to assigning –1 to *int-exp*. It refers to the last record in the object collection.

6. Using the NEXT clause is equivalent to using RELATIVE +1. It refers to the record in the object collection immediately after the current position.

7. Using the PRIOR clause is equivalent to using RELATIVE –1. It refers to the record in the object collection immediately before the current position.

8. If you use *int-exp* without the RELATIVE clause, the DBCS selects the record whose ordinal position in the object collection is equal to *int-exp*.

   - If *int-exp* is positive, the ordinal position is relative to the first record of the object collection.
   - If *int-exp* is negative, the ordinal position is relative to the last record of the object collection.
   - If *int-exp* is zero, a database exception condition occurs.
   - If *int-exp* is greater than the cardinality of the collection, a database exception condition occurs.

9. If you use the RELATIVE *int-exp* clause, the selected record is the one whose ordinal position in the object collection, relative to the position specified by the object currency indicator, is equal to *int-exp*.

   If the object currency indicator does not specify a position in the object collection, it is as though the RELATIVE clause did not appear.

   - If *int-exp* is positive, the search in the object collection proceeds in the next direction.
   - If *int-exp* is negative, the search in the object collection proceeds in the prior direction.
   - If *int-exp* is zero, a database exception condition occurs.

- If the current position, modified by *int-exp*, is zero or is greater in magnitude than the cardinality of the collection, a database exception condition occurs.

All Formats

The record selection expression causes a database exception condition to occur if:

1. The selected set currency indicator does not identify a record (DB-CONDITION is set to DBM $_CSTYP_NULL).
2. *Int-exp* is zero (DB-CONDITION is set to DBM$_BADZERO).
3. The program attempts to find or fetch a record following the last record in the selected collection (DB-CONDITION is set to DBM$_END).
4. The program attempts to find or fetch a record preceding the first record in the selected collection (DB-CONDITION is set to DBM$_END).
5. A realm required to carry out the record selection is not in ready mode (DB-CONDITION is set to DBM$_NOTIP).
6. The *bool-exp* clause is used and any specified data item is not included in the Subschema Record Description entry for the object record type.

See Section 4.8.3: Exception Conditions and the USE Statement statement for an explanation of DBM $_symbolic constants.

4.7. Set Membership Options and DML Verbs

The VSI COBOL data manipulation language (DML) verbs CONNECT, DISCONNECT, ERASE, MODIFY, RECONNECT, and STORE can affect a record's set membership. The effects of these verbs depend on the INSERTION and RETENTION clauses declared for the record's membership in each set in the schema.

The member's INSERTION clause determines whether the record is automatically inserted into a set when it is stored:

- If the INSERTION IS AUTOMATIC clause is used, the member is automatically inserted into the set.
- If the INSERTION IS MANUAL clause is used, the member must be manually inserted into the set with a CONNECT statement.

The member's RETENTION clause determines whether the record can be removed from a set with the verbs ERASE, DISCONNECT, and RECONNECT. If the RETENTION IS FIXED clause is used, you cannot remove the record from a set occurrence at all unless you erase the record at the same time. If the RETENTION IS MANDATORY clause is used, you cannot use DISCONNECT to remove the record from a set occurrence; you can use RECONNECT to move it from one occurrence of the set type to another. If the RETENTION IS OPTIONAL clause is used, you can use either DISCONNECT or RECONNECT to remove the record from a set occurrence.

The ERASE statement always removes the erased record from all sets of which it is a member. ERASE also affects sets owned by that record. If you use the ERASE ALL option, all members of sets owned by an erased record are erased in a recursive process. If you use ERASE without the ALL option, the

effect on each set member depends on the member's RETENTION clause: FIXED members are erased; OPTIONAL members are not erased (but are removed from the set, since the set is about to vanish). If any members have a RETENTION MANDATORY clause, an exception occurs because they can exist in the database without being members of this set occurrence. However, the Database Control System does not know into which set occurrence the members should be inserted.

The MODIFY statement may reposition a record within the same set occurrence if its sort key for that set is one of the data items being modified.

See the Oracle CODASYL DBMS DML documentation that summarizes the effects of the various verbs on the record directly modified and on any members (ERASE only).

4.8. Programming for Database Exceptions and Error Handling

Your program must contain logic to accommodate exceptions and errors. The items of syntax in VSI COBOL for OpenVMS that are used for this purpose are ON ERROR, AT END, and USE.

4.8.1. Database On Error Condition

The database *on error* exception condition occurs when the DBCS encounters a database exception condition for any data manipulation language (DML) statement.

The ON ERROR phrase in a DML statement allows the selection of an imperative statement sequence when any database exception condition occurs.

The NOT ON ERROR phrase allows execution of an imperative statement when a database exception condition does not occur.

The format is as follows:

[NOT] **ON ERROR stment**

stment is an imperative statement.

When a database exception condition occurs and the statement contains an ON ERROR phrase:

1. The imperative statement associated with the ON ERROR phrase executes.
2. The NOT ON ERROR phrase, if specified, is ignored.
3. Control is transferred to the end of database statement unless control has been transferred by executing the imperative statement of the ON ERROR phrase.

When a database exception condition occurs and the statement does not contain an ON ERROR phrase:

1. The applicable USE FOR DB-EXCEPTION, if specified, executes.
2. If an applicable USE procedure does not exist, the run unit terminates abnormally.
3. The NOT ON ERROR phrase, if specified, is ignored.

When a database exception condition does not occur:

1. The imperative statement associated with the NOT ON ERROR phrase, if specified, is executed.

2. The ON ERROR phrase, if specified, is ignored.

3. Control is transferred to the end of the database statement unless control has been transferred by executing the imperative statement of the NOT ON ERROR phrase.

Use the ON ERROR phrase to transfer execution control to the associated statements' error handling routine, where your program can supply useful and effective debugging information. (See Section 4.8.3 for examples and Glossary of Oracle DBMS-Related Terms for more information on Oracle CODASYL DBMS Database Special Registers.) The ON ERROR phrase can be part of every DML statement. It allows you to plan the graceful termination of a program that would otherwise terminate abnormally. (In a FETCH or FIND statement, you cannot specify both the ON ERROR and AT END phrases in the same statement.) For example:

```
PROCEDURE DIVISION.
    .
    .
    .

    RECONNECT PARTS_RECORD WITHIN ALL
              ON ERROR DISPLAY "Exception on RECONNECT"
                       PERFORM PROCESS-EXCEPTION.
    .
    .
    .
PROCESS-EXCEPTION.
    DISPLAY "Database Exception Condition Report".
    DISPLAY " ".
    DISPLAY "DB-CONDITION            = ", DB-CONDITION
                                          WITH CONVERSION.
    DISPLAY "DB-CURRENT-RECORD-NAME  = ", DB-CURRENT-RECORD-NAME.
    DISPLAY "DB-CURRENT-RECORD-ID    = ", DB-CURRENT-RECORD-ID
                                          WITH CONVERSION.
    DISPLAY " ".
    CALL "DBM$SIGNAL".
    STOP RUN.
```

4.8.2. At End Condition

An *at end condition* occurs when a program detects the end of a file. The at end condition may occur as a result of a FETCH or FIND statement execution in your database program. You use the AT END phrase to specify the action your program is to take when an at end condition occurs.

The NOT AT END phrase specifies the action your program takes if an AT END does occur.

Use the AT END phrase of the FETCH and FIND statements to handle the end of a collection of records condition. Your program will terminate if: (1) an at end condition occurs, (2) the program does not include the AT END phrase, and (3) there is no applicable USE statement.

When an at end condition occurs and the statement contains an AT END phrase:

1. The imperative statement associated with the AT END phrase, if specified, executes.

2. The NOT AT END phrase, if specified, is ignored.

3. Control is transferred to the end of the I/O statement unless control has been transferred by executing the imperative statement of the AT END phrase.

When an at end condition occurs and the statement does not contain an AT END phrase:

1. If the at end condition is associated with a FETCH or FIND statement, an applicable USE FOR DB-EXCEPTION procedure, if specified, executes. If the AT END phrase or USE FOR DB-EXCEPTION procedure is not specified, the run unit terminates abnormally.

2. If the at end condition is associated with a FETCH or FIND statement, DB-CONDITION is set to DBM$_END.

3. The NOT AT END phrase, if specified, is ignored.

When an at end condition (or any other error condition) does not occur:

1. The AT END phrase, if specified, is ignored.

2. The imperative statement associated with the NOT AT END phrase, if specified, is executed.

3. Control is transferred to the end of the I/O statement unless control has been transferred by executing the imperative statement of the NOT AT END phrase.

4.8.3. Exception Conditions and the USE Statement

Planning for exception conditions is an effective way to increase program and programmer productivity. A program with USE statements is more flexible than a program without them. They minimize operator intervention and often reduce or eliminate the time you need to debug and rerun the program.

The USE statement traps unsuccessful run-time Oracle CODASYL DBMS exception conditions that cause the execution of a Declaratives procedure. A Declaratives procedure can:

- Supply useful and effective database debugging information (see Section 2.2 for more information on Oracle CODASYL DBMS Database Special Registers)

- Provide alternate processing paths for specific exception conditions

Two sets of USE statements follow:

- The first set, shown in Example 4.1, consists of a single USE statement. This database USE procedure executes for any and all database exception conditions. If you select this set, it must be the only database USE statement in the Declaratives Section. Its format is:

  ```
  USE [ GLOBAL ] FOR DB-EXCEPTION.
  ```

  Example 4.1. A Single USE Statement

  ```
  PROCEDURE DIVISION.
  DECLARATIVES.
  200-DATABASE-EXCEPTIONS SECTION. USE FOR DB-EXCEPTION.
  DB-ERROR-ROUTINE.
      DISPLAY "Database Exception Condition Report".
      DISPLAY "-----------------------------------------".
      DISPLAY "DB-CONDITION            = ", DB-CONDITION
                                           WITH CONVERSION.
      DISPLAY "DB-CUR-REC-NAME  = ", DB-CURRENT-RECORD-NAME.
      DISPLAY "DB-CURRENT-RECORD-ID    = ", DB-CURRENT-RECORD-ID
                                                 WITH CONVERSION.
      DISPLAY "DB-CUR-REC-ID    = ", DB-CRID.
  ```

```
    DISPLAY " ".
    CALL "DBM$SIGNAL".
END DECLARATIVES.
```

- The second set, shown in Example 4.2, consists of one or more Format 1 USE statements, and one Format 2 USE statement.

Format 1

```
USE [GLOBAL] FOR DB-EXCEPTION ON DBM$_exception-condition [, DBM
$_exception-condition]...
```

A Format 1 database declarative executes whenever a database exception condition occurs and the corresponding DBM$_exception-condition is explicitly stated in the USE statement.

Format 2

```
USE [ GLOBAL ] FOR DB-EXCEPTION ON OTHER.
```

A Format 2 declarative executes whenever a database exception condition occurs and the corresponding DBM$_exception-condition is not explicitly stated in any Format 1 USE statement.

Example 4.2. Multiple USE Statements

```
PROCEDURE DIVISION.
DECLARATIVES.
200-DATABASE-EXCEPTIONS SECTION.
    USE FOR DB-EXCEPTION ON DBM$_CRELM_NULL,
                            DBM$_CRTYPE_NULL.
200-DATABASE.
    PERFORM 300-REPORT-DATABASE-EXCEPTIONS.
    IF DB-CONDITION = ..... GO TO ...
    IF DB-CONDITION = ..... GO TO ...
    STOP RUN.
225-DATABASE-EXCEPTIONS SECTION.
    USE FOR DB-EXCEPTION ON DBM$_DUPNOTALL.
225-DATABASE.
    PERFORM 300-REPORT-DATABASE-EXCEPTIONS.
    GO TO ...
250-DATABASE-EXCEPTIONS SECTION.
    USE FOR DB-EXCEPTION ON OTHER.
250-DATABASE.
    PERFORM 300-REPORT-DATABASE-EXCEPTIONS.
    EVALUATE DB-CONDITION
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN .....              GO TO ...
             WHEN OTHER              PERFORM... .
    STOP RUN.
300-REPORT-DATABASE-EXCEPTIONS.
    DISPLAY "Database Exception Condition Report".
```

```
DISPLAY " ".
DISPLAY "DB-CONDITION     = ",     DB-CONDITION
                                   WITH CONVERSION.
DISPLAY "DB-CUR-REC-NAME  = ",     DB-CURRENT-RECORD-NAME.
DISPLAY "DB-CURRENT-RECORD-ID = ", DB-CURRENT-RECORD-ID
DISPLAY " ".
CALL "DBM$SIGNAL".
```

4.8.4. Translating DB-CONDITION Values to Exception Messages

Oracle CODASYL DBMS includes the following procedure for exception condition handling:

```
CALL "DBM$SIGNAL".
```

Use this procedure when it is necessary to output an exception message rather than, or in addition to, displaying the numeric value of DB-CONDITION. For more information on the Oracle CODASYL DBMS database special register DB-CONDITION, see Section 2.2.1.

4.9. Database Programming Statements in the COBOL Procedure Division

The VSI COBOL for OpenVMS Oracle CODASYL DBMS DML statements are:

- COMMIT
- CONNECT
- DISCONNECT
- ERASE
- FETCH
- FIND
- FREE
- GET
- KEEP
- MODIFY
- READY
- RECONNECT
- ROLLBACK
- STORE
- USE (Format 3)

- The RETAINING clause

COMMIT

COMMIT — Ends your database transaction, makes permanent all changes made to the database since the last quiet point, and establishes a new quiet point for this run unit.

General Format

COMMIT [STREAM stream name] [RETAINING {retaining list}]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-COMMIT]

stment
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

Syntax Rules

The STREAM clause cannot be specified if the subschema entry (DB) does not name a stream.

General Rules

1. The COMMIT statement ends the database transaction.
2. If you do not specify the STREAM clause, the COMMIT statement makes all changes made to the database since the last quiet point for this run unit; a new quiet point is established for the run unit.
3. If you specify the STREAM clause, the COMMIT statement makes all changes made to the database since the last quiet point permanent for the specified stream; a new quiet point is then established for the stream.
4. If you do not use the RETAINING clause (see Section 4.10: RETAINING Clause), COMMIT:
   - Empties all keeplists
   - Resets all your currency indicators to null
   - Releases all realm and record locks
   - Makes visible any changes you made to the database
   - Terminates the READY mode for each target realm
5. If you do use the RETAINING clause, COMMIT:
   - Does not empty keeplists

- Retains all currency indicators
- Does not release realm locks
- Demotes no-read record locks to read-only record locks, then releases locks for all records except those in currency indicators or keeplists
- Makes visible any changes you made to the database
- Maintains READY modes for each target realm

6. If a database exception condition occurs during the execution of a COMMIT statement, the DBCS places a database exception condition code in the special register DB-CONDITION. This code identifies the condition.

Additional References

- Section 2.2 on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- USE statement (Format 3)
- Section 5.2.4, on subschema description (DB)

Example

```
COMMIT ON ERROR PERFORM 200-DISPLAY-ERROR-ON-COMMIT
                END-COMMIT.
```

CONNECT

CONNECT — Inserts the current record of the run unit as a member record into one or more sets. The set occurrence for each insertion is determined by the currency indicator for the corresponding set type.

General Format

CONNECT [record-name] **TO** { {set-name}... | ALL }

[RETAINING [{ { REALM | RECORD | { SET [set-name]... | { set-name }... } } }] CURRENCY]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-CONNECT]

record-name
names a subschema record type.

set-name

names a subschema set type.

`stment`
is an imperative statement executed for an on error condition.

`stment2`
is an imperative statement executed for a not on error condition.

Syntax Rules

1. *Record-name* must reference a subschema record type.
2. *Set-name* must reference a subschema set type.
3. For each *set-name* in the TO *set-name* clause, the record type of *record-name* must be a member record type of the set type.
4. The same *set-name* cannot be specified more than once in the TO *set-name* clause.
5. The same *set-name* cannot be specified more than once in the RETAINING clause.

General Rules

1. The current record of the run unit must be in a realm in ready update mode and all records required by the Database Control System (DBCS) to execute the CONNECT statement must be in a realm in available mode. The CONNECT statement uses the current record of the run unit.
2. Use *record-name* to check that the current record of the run unit is a record type identical to the *record-name* record type.
3. The current record of the run unit must not already be a member of *set-name*.
4. For each *set-name* in the TO clause:
   a. The DBCS inserts the current record of the run unit into each *set-name* as determined by the set currency indicator associated with each *set-name*.
   b. The position where the DBCS inserts the record into the set is determined according to the criteria for ordering the set defined in the schema for *set-name*.
5. When specifying the ALL option:
   a. The DBCS considers only those set types specified in your subschema for which:
      - The current record is not presently a member of any occurrence of the set type.
      - The record type of the current record is defined in the schema as a member record type of the set type.
   b. For each such selected set type:
      - The DBCS inserts the current record into the set determined by the set currency indicator associated with the selected set type.
      - The position where the DBCS inserts the current record into the set is determined according to the set-ordering criteria in the schema for the selected set type.

6. Unless otherwise specified by the RETAINING clause (see Section 4.10: RETAINING Clause), set type currency indicators for the connected sets point to the connected record. All other currency indicators are not affected.

7. If a database exception condition occurs during the execution of a CONNECT statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Notes). This code identifies the condition.

8. If the execution of a CONNECT statement results in a database exception condition, no changes are made to the membership of the current record in the database.

Technical Notes

CONNECT statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_CRUN_NULL	The currency indicator for the run unit is null.
DBM$_CRUN_POS	The currency indicator for the run unit specifies the position of a vacated record in a record collection.
DBM$_WRONGRTYP	The record type of *record-name* is not the same as the current record's type.
DBM$_NOT_MTYP	The current record is not a member record type of a set in the TO *set-name* phrase.
DBM$_NOW_MBR	The current record is already a member of the set specified in the TO *set-name* phrase.
DBM$_CSTYP_NULL	There is no current of set type for the set specified in the TO *set-name* phrase. This occurs only if the set is not a singular set.
DBM$_DUPNOTALL	The program attempts to connect a record to a set and its sort key value is identical to another record's sort key value already in the set.
DBM$_NOT_UPDATE	A realm is not in update usage mode.
DBM$_CHKMEMBER	The Oracle CODASYL DBMS CHECK (member) condition was evaluated to be false. The database remains unchanged.

Additional References

- Section 2.2 on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- Section 5.14.1, on RETAINING clause
- USE statement

Examples

1. Connecting EMPLOYEE record to the CONSISTS_OF set:

```
CONNECT EMPLOYEE TO CONSISTS_OF.
```

2. Connecting EMPLOYEE record to all sets:

```
CONNECT EMPLOYEE TO ALL.
```

3. Connecting EMPLOYEE record to the CONSISTS_OF set without changing the currency indicator of the CONSISTS_OF set:

```
CONNECT EMPLOYEE TO CONSISTS_OF
        RETAINING SET CONSISTS_OF CURRENCY.
```

DISCONNECT

DISCONNECT — Removes the current record of the run unit from one or more sets.

General Format

DISCONNECT [record-name] **FROM** { { set-name }... | ALL }

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-DISCONNECT]

record-name
names a subschema record type.

set-name
names a subschema set type.

stment
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

Syntax Rules

1. The record type of *record-name* must be a member record type of the set type for each *set-name*.
2. The same *set-name* cannot be specified more than once in the same DISCONNECT statement.

General Rules

1. The DISCONNECT statement references the current record of the run unit.
2. The current record of the run unit must be in a realm in ready mode and all records required by the DBCS to execute the DISCONNECT statement must be in a realm in available mode.
3. Use *record-name* to check that the current record of the run unit has the same record type as *record-name*.
4. The current record of the run unit must be an OPTIONAL member of each *set-name*.
5. The record type of the current record of the run unit must be a member record type of the set type for each *set-name*.

6. The DBCS removes the current record of the run unit from each *set-name*.
7. Use the ALL clause to remove the current record of the run unit from as many sets as possible. The DBCS considers only those set types defined in the subschema for which the record is an OPTIONAL member. Set types not included in your subschema and set types of which the record is not a member are ignored. The current record of the run unit is removed from each remaining set type.
8. If the set type currency indicator for a disconnected set pointed to the current record, that currency indicator now points to the position in the set vacated by the record. All other currency indicators are not affected.
9. If the execution of a DISCONNECT statement results in a database exception condition, no changes are made to the membership of the current record in the database.
10. If a database exception condition occurs during the execution of a DISCONNECT statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Notes). This code identifies the condition.

Technical Notes

DISCONNECT statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_NOT_OPTNL	The current record of the run unit is not an OPTIONAL member of set-name.
DBM$_CRUN_NULL	The currency indicator for the current record of the run unit is null.
DBM$_CRUN_POS	The currency indicator for the current record of the run unit specifies the position of a vacated record in a record collection.
DBM$_WRONGRTYP	The record type of record-name is not the same as the current record's type.
DBM$_NOT_MTYP	The current record is not a member record type of set-name.
DBM$_NOT_MBR	The current record is not a member of set-name.
DBM$_NOT_UPDATE	A realm is not in ready update usage mode.

Additional References

- Section 2.2 on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- USE statement

Examples

```
DISCONNECT EMPLOYEE FROM CONSISTS_OF.
DISCONNECT EMPLOYEE FROM ALL.
DISCONNECT FROM ALL.
DISCONNECT FROM CONSISTS_OF.
```

ERASE

ERASE — Deletes the current record of the run unit from the database. Additional records owned by the current record may also be deleted and/or disconnected.

General Format

ERASE [ALL] [record-name]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-ERASE]

`record-name`
names a subschema record type.

`stment`
is an imperative statement executed for an on error condition.

`stment2`
is an imperative statement executed for a not on error condition.

General Rules

1. The ERASE statement references the current record of the run unit.
2. Use *record-name* to check that the current record of the run unit has the same record type as *record-name*.
3. The current record of the run unit must be in a realm in update ready mode, and all records required by the Database Control System (DBCS) to execute the ERASE statement must be in a realm in available mode.
4. The current record of the run unit is called the *object record* for the remaining General Rules.
5. The object record is erased from all sets in which it is a member and deleted from the database.
6. If you use the ALL phrase, the object record and all sets owned by the object record are erased (independent of set membership retention class).
7. If you do not use the ALL phrase, these rules apply to the members of the sets owned by the erased object record:
   - Each member with FIXED membership is erased.
   - Each member with OPTIONAL membership is disconnected from the set but not deleted from the database.
   - An exception occurs if any members have MANDATORY membership.
8. The database key value corresponding to the erased object record is removed from all keeplists.
9. General Rules 5 to 8 apply to each erased record as if it were the object record.

10. The successful ERASE statement changes currency indicators as follows:
    - The currency indicator for the current record of the run unit points to the position in the database vacated by the current record of the run unit. However, you no longer have a current record.
    - If an erased record is the current record of a disconnected set type, that set type currency indicator now points to the position in the set vacated by that record.
    - If an erased record is the owner of the current set of any set type, that set type's currency indicator becomes null.
    - If an erased record is the current record of a record type, that record type currency indicator becomes null.
    - If the erased record is the current record of the realm, that realm currency indicator now points to the position in the realm vacated by that record.
11. If the execution of an ERASE statement results in a database exception condition, no changes are made to the database.
12. If a database exception condition occurs during the execution of an ERASE statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Notes). This code identifies the condition.

Technical Notes

ERASE statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_CRUN_NULL	The currency indicator for the run unit is null.
DBM$_CRUN_POS	The currency indicator for the run unit specifies the position of a vacated record in the record collection.
DBM$_WRONGRTYP	The record type of record-name is not the same as the current record's type.
DBM$_ERASEMANDT	You attempted to erase a record that is the owner of a set occurrence with a MANDATORY member record occurrence, but you did not use the ALL option.
DBM$_NOT_UPDATE	A realm is not in update usage mode.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- USE statement

Examples

1. This example erases PART record, all records owned by PART in the PART_USES, PART_INFO, and PART_USED_ON sets. It also erases all SUPPLY and PR_QUOTE records that it owns through the PART_INFO set. It disconnects PART from the RESPONSIBLE_FOR set.

```
ERASE PART.
```

2. This example has the same effect as in example 1 for each PART record that the CLASS1 record owns; however, it also erases the CLASS1 record.

```
ERASE ALL CLASS1.
```

Note

Because CLASS1 has MANDATORY members, you *cannot* erase only CLASS1.

FETCH

FETCH — Is a combined FIND and GET that establishes a specific record in the database as the current record of the run unit and makes the record available to the run unit in the user work area.

General Format

FETCH database-record

[FOR UPDATE]

[RETAINING [{ { REALM | RECORD | { SET [set-name]... | {set-name}... } } }] CURRENCY]

[{ [AT END stment] [NOT AT END stment2] | [ON ERROR stment] [NOT ON ERROR stment2] }]

[END-FETCH]

database-record
represents a record selection expression. References are made to a record in the database according to the rules for Record Selection Expressions (see Section 4.6).

set-name
names a subschema set type.

stment
is an imperative statement executed for an AT END or ON ERROR condition.

stment2
is an imperative statement executed for a NOT AT END or NOT ON ERROR condition.

General Rules

1. *database-record* must reference a record stored in a realm that is in ready mode.

2. The FETCH statement is equivalent to the following sequence of statements:

```
FIND database-record . . .
GET . . .
```

3. Execution of the FETCH statement causes *database-record* to become the current record of the run unit.

4. The FOR UPDATE option puts a no-read lock on the specified record.

5. The DBCS makes a copy of the selected database record available to the program in the user work area. Any change made to the user work area does not affect the record in the database. You must execute a MODIFY and COMMIT statement to make permanent changes to the database.

6. Unless otherwise specified by the RETAINING clause (see Section 4.10: RETAINING Clause), the successful FETCH statement causes the DBCS to update these currency indicators to point to the selected record:

   - Run unit.
   - Realm.
   - Record type.
   - Set type. If it is an owner record type of the set type, this currency indicator points to the selected record even if the set is empty. If it is a member record type of the set type, this currency indicator points to the selected record only if it is currently a member of some set of that record type.

7. The DBCS places the database key of the fetched record in the special register DB-KEY.

8. If a database exception condition occurs during the execution of a FETCH statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Note). This code identifies the condition.

9. [NOT]ON ERROR and [NOT]AT END cannot be used concurrently in a Fetch statement.

10. If either AT END or ON ERROR is used, it must precede USE.

11. If ON ERROR and NOT ON ERROR are used concurrently in a Fetch statement, USE procedures will not be activated.

Technical Note

FETCH statement execution can result in these database exception conditions and those associated with the evaluation of the record selection expression:

DBM$_CONVERR	A data conversion error occurred in the FETCH operation.
DBM$_ILLNCHAR	Invalid character found in a numeric field.
DBM$_NONDIGIT	Nonnumeric character found in a numeric field.
DBM$_OVERFLOW	A data overflow error occurred in the FETCH operation.
DBM$_TRUNCATION	A data truncation error occurred in the FETCH operation.
DBM$_UNDERFLOW	A data underflow error occurred in the FETCH operation.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements

- Section 4.6, on record selection expressions
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on AT END phrase
- Section 4.8.1, on database On Error condition
- Section 5.14.1, on RETAINING clause
- USE statement

Examples

1. Currency indicator access. To retrieve the:
   - Current record of the run unit

     ```
     FETCH CURRENT.
     ```

   - Current record of the MAKE realm

     ```
     FETCH CURRENT WITHIN MAKE.
     ```

   - Current record of the CLASS_PART set

     ```
     FETCH CURRENT WITHIN CLASS_PART.
     ```

   - Current PART record

     ```
     FETCH CURRENT WITHIN PART.
     ```

2. Keeplist access. To retrieve the:
   - First entry in the keeplist

     ```
     FETCH FIRST WITHIN KEEPLIST-A.
     ```

   - Last entry in the keeplist

     ```
     FETCH LAST WITHIN KEEPLIST-B.
     ```

   - N(th) entry in the keeplist

     ```
     FETCH OFFSET RECORD-COUNT WITHIN KEEPLIST-C.
     ```

3. Set owner access. To retrieve the owner (PART) of the PART_USES set:

   ```
   FETCH OWNER WITHIN PART_USES.
   ```

4. Record search access. To retrieve the:
   - First member within the PART_USES set

     ```
     FETCH FIRST WITHIN PART_USES.
     ```

   - First part whose PART-COST is greater than 300 dollars

     ```
     FETCH FIRST PART WHERE PARTCOST GREATER THAN 300
     ```

   - Last member within the BUY realm

```
FETCH LAST WITHIN BUY.
```

- Next COMPONENT record in the PART_USED_ON set

```
FETCH NEXT COMPONENT WITHIN PART_USED_ON.
```

- Prior COMPONENT record in the PART_USES set using COMP_SUB_PART and COMP_OWNER_PART as record keys

```
FETCH PRIOR COMPONENT WITHIN PART_USES
      USING COMP_SUB_PART, COMP_OWNER_PART.
```

- Record relative to the current record, this statement starts from the current position to the position defined by this offset

```
FETCH RELATIVE SEARCH-NUMBER
                  ON ERROR PERFORM 300-FETCH-IN-ERROR-ROUTINE
                  END-FETCH.
```

- PART record with PART_STATUS equal to “G” from the CLASS_PART set

```
MOVE "G" TO PART_STATUS.
FETCH NEXT CLASS WITHIN ALL_CLASS.
FETCH NEXT PART WITHIN CLASS_PART USING PART_STATUS
                AT END . . .
```

- DB-KEY access. To retrieve the item referred to by the DB-KEY special register:

```
FETCH DBKEY.
```

FIND

FIND — Locates a specific record in the database and establishes it as the current record of the run unit. The FIND ALL statement locates zero or more records in the database and inserts them into the named keeplist.

General Format

Format 1

FIND database-record [FOR UPDATE]

[RETAINING [{ { RETAINING | RECORD | { SET [set-name]... | {set-name}... } } }] CURRENCY]

[{ [AT END stment] [NOT AT END stment2] | [ON ERROR stment] [NOT ON ERROR stment2] }]

[END-FIND]

Format 2

FIND ALL keeplist-name [record-name] [WITHIN { realm-name | set-name }]

[USING{rec-key}... | WHERE{bool-expres}] [FOR UPDATE]

[{ [AT END stment] [NOT AT END stment2] | [ON ERROR stment] [NOT ON ERROR stment2] }]

[END-FIND]

`database-record`
represents a record selection expression. References are made to a record in the database according to the rules for Record Selection Expressions (see Section 4.6).

`set-name`
names a subschema set type.

`stment`
is an imperative statement executed for an at end or on error condition.

`stment2`
is an imperative statement executed for a not at end or not on error condition.

`keeplist-name`
names a list of database keys used by a run unit to lock records for later reference.

`record-name`
names a subschema record type.

`realm-name`
names a subschema realm.

`rec-key`
is a key data item within the subschema record occurrence.

`bool-expres`
is a conditional expression that involves data items of the object record.

General Rules

Format 1

1. *database-record* must reference a record stored in a realm that is in ready mode.
2. Execution of the FIND statement causes *database-record* to become the current record of the run unit.
3. The Database Control System (DBCS) does not make a copy of the database record available to the program. To retrieve the record you must use a FETCH or GET statement.
4. The FOR UPDATE option puts a no-read lock on the specified record.
5. Unless otherwise specified by the RETAINING clause (see Section 4.10: RETAINING Clause), the successful FIND statement causes the DBCS to update these currency indicators to point to the selected record:
   - Run unit.
   - Realm.
   - Record type.

- Set type. If the record is an owner record of the set type, this currency indicator points to the selected record even if the set is empty. If the record is a member record of the set type, this currency indicator points to the selected record only if it is currently a member of some set of that record type.

Format 2

1. *keeplist-name* identifies the object keeplist for the statement.
2. Using FIND ALL puts the database key values corresponding to each qualifying record at the end of the object keeplist.
3. When FIND ALL *keeplist-name* is used, no currency indicators are affected.
4. The same *rec-key* can appear only once in a given USING phrase.
5. The *bool-expres* expression is used to specify additional requirements of a qualifying record.
6. When FOR UPDATE is used, no-read locks may be placed on more than one record.

All Formats

1. If a database exception condition occurs during the execution of a FIND statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Note). This code identifies the condition.
2. The DBCS places the database key of the record found in the special register DB-KEY.
3. [NOT]ON ERROR and [NOT]AT END cannot be used concurrently in a Find statement.
4. If either AT END or ON ERROR is used, it must precede USE.
5. If ON ERROR and NOT ON ERROR are used concurrently in a Find statement, USE procedures will not be activated.

Technical Note

FIND statement execution can result in the database exception conditions associated with the evaluation of the record selection expression.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on AT END phrase
- Section 4.6, on record selection expressions
- Section 4.8.1, on database On Error condition
- Section 5.14.1, on RETAINING clause

- USE statement

Examples

1. Currency indicator access. To locate the following items, use the statement listed with each item:
   - Current record of the run unit

     ```
     FIND CURRENT.
     ```

   - Current record of the MAKE realm

     ```
     FIND CURRENT WITHIN MAKE.
     ```

   - Current record of the CLASS_PART set

     ```
     FIND CURRENT WITHIN CLASS_PART.
     ```

   - Current PART record

     ```
     FIND CURRENT WITHIN PART.
     ```

2. Keeplist access. To locate the following items, use the statement listed with each item:
   - (ALL) PART records with a PARTSTATUS OF H and put their dbkey values in keeplist THREE

     ```
     FIND ALL THREE PART USING PARTSTATUS PARTSTATUS [X(1)] = H
     ```

   - First entry in the keeplist

     ```
     FIND FIRST WITHIN KEEPLIST-A.
     ```

   - Last entry in the keeplist

     ```
     FIND LAST WITHIN KEEPLIST-B.
     ```

   - N(th) entry in the keeplist

     ```
     FIND OFFSET RECORD-COUNT WITHIN KEEPLIST-C.
     ```

3. Set owner access. To locate the owner (PART) of the PART_USES set:

   ```
   FIND OWNER WITHIN PART_USES.
   ```

4. Record search access. To locate the following items, use the statement listed with each item:
   - NEXT DEALER record whose DEAL-PHONE item begins with the area code numbers 617

     ```
     FIND NEXT DEALER WHERE DEAL-PHONE MATCHES "617"
     ```

   - First member within the PART_USES set

     ```
     FIND FIRST WITHIN PART_USES.
     ```

   - Last member within the BUY realm

     ```
     FIND LAST WITHIN BUY.
     ```

   - Next COMPONENT record in the PART_USED_ON set

```
FIND NEXT COMPONENT WITHIN PART_USED_ON.
```

- Prior COMPONENT record in the PART_USES set (uses COMP_SUB_PART and COMP_OWNER_PART as record keys)

```
FIND PRIOR COMPONENT WITHIN PART_USES
       USING COMP_SUB_PART, COMP_OWNER_PART.
```

- A record relative to the current record (this statement starts from the current position to the position defined by this offset)

```
FIND RELATIVE SEARCH-NUMBER
      ON ERROR PERFORM 300-FIND-IN-ERROR-ROUTINE
      END-FIND.
```

- PART record with PART_STATUS equal to “G” from the CLASS_PART set

```
MOVE "G" TO PART_STATUS.
FIND NEXT CLASS WITHIN ALL_CLASS.
FIND NEXT PART WITHIN CLASS_PART USING PART_STATUS
                 AT END...
```

- All PART records with PART_STATUS OF G (this statement puts their dbkey values in KEEPLIST-4)

```
FIND ALL KEEPLIST-4 PART USING PART_STATUS
```

- DB-KEY access. To locate the item referred to by the DB-KEY special register:

```
FIND DBKEY.
```

FREE

FREE — Empties selected keeplists or removes a database key value from a keeplist or currency indicator.

General Format

FREE { database-key-id | ALL [{ FROM {keeplist-name}... | CURRENT }] }

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-FREE]

`database-key-id`
represents a database key identifier. References are made to a keeplist entry or a currency indicator according to the rules for Database Key Identifiers.

`keeplist-name`
names a keeplist in the Subschema Section.

`stment`
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

Syntax Rule

Keeplist-name cannot be specified more than once in a given FREE statement.

General Rules

1. If *database-key-id* references a keeplist entry, the DBCS deletes that keeplist entry from the keeplist containing it. Removing an entry from a keeplist changes the position of all subsequent entries in the keeplist.
2. If *database-key-id* references a currency indicator, the DBCS sets that currency indicator to null.
3. If ALL is specified and the FROM phrase is omitted, the DBCS empties all keeplists in the program.
4. If ALL CURRENT is specified, the DBCS nulls all of your currencies. It does not free entries in a keeplist.
5. If the FROM phrase is specified, the DBCS empties each *keeplist-name*.
6. The FREE statement releases retrieval update locks on the target record.
7. If a database exception condition occurs during the execution of a FREE statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Note). This code identifies the condition.

Technical Note

FREE statement execution can result in these database exception conditions and those associated with a Database Key Identifier:

DBM$_BADZERO	*Integer-exp* is zero.
DBM$_END	You accessed a keeplist with fewer entries than you expected.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.4, on database key identifiers
- Section 4.8.1, on database On Error condition
- USE statement

Examples

1. Currency indicator access
   - To free the current record of the run unit:

```
FREE CURRENT.
```

- To free the currency indicator for the PART record:

```
FREE CURRENT WITHIN PART.
```

- To free all currencies:

```
FREE ALL CURRENT.
```

2. Keeplist access

- To free the first keeplist entry in KEEPLIST-A:

```
FREE FIRST WITHIN KEEPLIST-A.
```

- To free the last keeplist entry in KEEPLIST-A:

```
FREE LAST WITHIN KEEPLIST-A.
```

- To free the second keeplist entry relative to the current position in the keeplist:

```
FREE OFFSET 2 WITHIN KEEPLIST-B.
```

- To empty a keeplist:

```
FREE ALL FROM KEEPLIST-B
```

- To empty all keeplists:

```
FREE ALL
```

GET

GET — Moves the contents of the current database record of the run unit to your user work area.

General Format

GET [record-name | {record-item}...]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-GET]

record-name
names a subschema record type.

record-item
is a group or elementary data item in a subschema record type. Record-item may be qualified.

stment
is an imperative statement executed for an on error condition.

stment2

is an imperative statement executed for a not on error condition.

Syntax Rules

1. Each *record-item* must reference a data item in the same subschema record type.
2. If *record-item* is a group item, its subordinate group and elementary items cannot be referenced in the same GET statement.

General Rules

1. The GET statement references the current record of the run unit.
2. If you use the *record-name* option, or if you do not specify either the *record-item* or the *record-name* option, the DBCS makes a copy of the entire current record of the run unit available to the program in the user work area.
3. If you use the *record-item* option, only those items in the current record of the run unit are made available to the program in the user work area.
4. Any change made to the user work area does not affect the record in the database. You must execute a MODIFY and a COMMIT statement to make permanent changes to the database.
5. Use *record-name* to check that the record type of the current record is identical to the record type specified by *record-name*.
6. The GET statement has no effect on currency indicators.
7. If a database exception condition occurs during the execution of a GET statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Note). This code identifies the condition.

Technical Note

GET statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_CRUN_NULL	The currency indicator for the run unit is null.
DBM$_CRUN_POS	The currency indicator for the run unit points to a vacated position in the database.
DBM$_WRONGRTYP	The record type for record-name is not identical to the current record type, or a record-item is not a group or elementary item in the current record.
DBM$_CONVERR	A data conversion error occurred in the GET operation.
DBM$_ILLNCHAR	An invalid character was found in a numeric field.
DBM$_NONDIGIT	A nonnumeric character was found in a numeric field.
DBM$_OVERFLOW	A data overflow error occurred in the GET operation.
DBM$_TRUNCATION	A data truncation error occurred in the GET operation.
DBM$_UNDERFLOW	A data underflow error occurred in the GET operation.

Additional References

- Section 2.2, on reserved words (database special registers)

- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- USE statement

Examples

1. This example gets the current record of the run unit:

   ```
   GET.
   ```

2. This example gets a PART record only if it is the current record of the run unit:

   ```
   GET PART.
   ```

3. This example gets these items only if they are part of the current record of the run unit:

   ```
   GET PART_ID, PART_DESC.
   ```

KEEP

KEEP — Inserts a database key value from a currency indicator or keeplist into a keeplist.

General Format

KEEP [database-key-id] **USING destination-keeplist**

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-KEEP]

database-key-id
represents a database key identifier. References are made to a keeplist entry or a currency indicator according to the rules for Database Key Identifiers.

destination-keeplist
names a keeplist in the Subschema Section to receive the database key value.

stment
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

General Rules

1. If *database-key-id* references a database key value in either a currency indicator or a keeplist, the DBCS:
   - Inserts that value as the last entry in *destination-keeplist*
   - Sets a lock on the associated record to prevent concurrent run units from accessing it

2. If the program does not specify *database-key-id*, the DBCS inserts the database key value of the current record of the run unit as the last entry of *destination-keeplist*.

3. The KEEP statement has no effect on currency indicators.

4. If a database exception condition occurs during the execution of a KEEP statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Note). This code identifies the condition.

Technical Note

KEEP statement execution can result in the database exception conditions associated with the evaluation of the database key identifier.

Additional References

- Section 2.2, on reserved words (database special registers)
- Keeplist Description entry
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.4, on database key identifiers
- Section 4.8.1, on database On Error condition
- USE statement

Examples

1. Currency indicator access

   - To insert the database key value of the current record of the run unit into KEEPLIST-A:

     ```
     KEEP CURRENT USING KEEPLIST-A.
     ```

   - To insert the database key value of the current record of the CLASS_PART set into KEEPLIST-B:

     ```
     KEEP CURRENT WITHIN CLASS_PART USING KEEPLIST-B.
     ```

   - To insert the database key value of the current record of the BUY realm into KEEPLIST-B:

     ```
     KEEP CURRENT WITHIN BUY USING KEEPLIST-B.
     ```

   - To insert the database key value of PART record into KEEPLIST-B:

     ```
     KEEP CURRENT WITHIN PART USING KEEPLIST-B.
     ```

2. Keeplist access

   - To insert the first keeplist entry from KEEPLIST-A as the last entry in KEEPLIST-B:

     ```
     KEEP FIRST WITHIN KEEPLIST-A USING KEEPLIST-B.
     ```

   - To insert the last keeplist entry in KEEPLIST-A as the last entry in KEEPLIST-B:

```
KEEP LAST WITHIN KEEPLIST-A USING KEEPLIST-B.
```

- To insert the fourth keeplist entry relative to the current position in KEEPLIST-A as the last entry in KEEPLIST-B:

```
KEEP OFFSET 4 WITHIN KEEPLIST-A USING KEEPLIST-B.
```

MODIFY

MODIFY — Changes the contents of specified data items in a database record.

General Format

MODIFY [record-name | {record-item}...]

[RETAINING [{ { REALM | RECORD | { SET[set-name]... | {set-name}... } } }] CURRENCY]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-MODIFY]

record-name
names a subschema record type.

record-item
is a group or elementary data item in a subschema record type. *Record-item* can be qualified.

set-name
names a subschema set type.

stment
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

Syntax Rules

1. Each *record-item* must reference a data item in the same subschema record type.
2. If *record-item* is a group item, its subordinate group and elementary data items cannot be referenced as a *record-item* in the same MODIFY statement.
3. Use *record-name* to check that the current record of the run unit is a record type identical to the *record-name* record type.

General Rules

1. Changes are made to a database record by:
   - Locating the target record (FIND and GET, or FETCH)
   - Moving the new data to the data-name references in the user work area

- Executing the MODIFY statement

2. The DBCS changes the contents of the entire database record:
   - If you specify *record-name*
   - If you specify neither *record-name* nor *record-item*
3. Using *record-item* limits the changes to the database record to only those items you specify.
4. If you modify any data item that is the sort key for that record's position in a sorted set, the DBCS automatically repositions the record in that set in accordance with the set-ordering criteria of the set type.

   If the value of the sort key is identical to another record of this type, and if duplicates are not allowed for this sorted set, the DBCS does not perform the MODIFY operation, and a database exception condition occurs.
5. Currency indicator update rules for record insertion and removal are as follows:
   - If the MODIFY statement removes a record from the current set, the currency indicator for that set points to the position in the set vacated by the record. (At this position the record is no longer accessible.)
   - If the MODIFY statement reinserts a record in a set in another location, the set currency indicator changes to point to the record in this new location.
   - If you do not include the RETAINING clause (see Section 4.10: RETAINING Clause), the currency indicator for the set is modified to the current record of the run unit.
6. The current record of the run unit must be in a realm in ready update mode and all records required by the DBCS to execute the MODIFY statement must be in realms in one of the appropriate READY modes.
7. The contents of the user work area (UWA) do not change after the successful or unsuccessful execution of a MODIFY statement.
8. If the execution of a MODIFY statement results in an exception condition, the record is not changed.
9. If a database exception condition occurs during the execution of a MODIFY statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Notes). This code identifies the condition.

Technical Notes

MODIFY statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_CRUN_NULL	The currency indicator for the run unit is null.
DBM$_CRUN_POS	The currency indicator for the run unit specifies the position of a vacated record in a record collection.
DBM$_WRONGRTYP	The record type of record-name is not the same type as the current record of the run unit, or the record-item is not a group or elementary item of the current record type.

DBM$_DUPNOTALL	A sort key data item is modified creating a duplicate value in a DUPLICATES NOT ALLOWED set.
DBM$_NOT_UPDATE	A realm is not in ready update usage mode.
DBM$_CHKITEM	A modified data item failed to pass the schema CHECK ITEM condition.
DBM$_CHKMEMBER	A modified data item failed to pass the schema CHECK MEMBER condition.
DBM$_CHKRECORD	A modified data item failed to pass the schema CHECK RECORD condition.
DBM$_CONVERR	A data conversion error occurred in the MODIFY operation.
DBM$_ILLNCHAR	An invalid character was found in a numeric field.
DBM$_NONDIGIT	A nonnumeric character was found in a numeric field.
DBM$_OVERFLOW	A data overflow error occurred in the MODIFY operation.
DBM$_TRUNCATION	A data truncation error occurred in the MODIFY operation.
DBM$_UNDERFLOW	A data underflow error occurred in the MODIFY operation.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- Section 5.14.1, on RETAINING clause
- USE statement

Examples

1. This example modifies an entire database record. It is a partial update routine. Except for the PART_ID number in PARTS, the set was originally created with empty parts records. As new information for a part is made available, it is included in the record.

```
150-UPDATE-PARTS-RECORD.

    DISPLAY "ENTER A PART-ID NUMBER - X(8): "
             WITHOUT ADVANCING.
    ACCEPT PART_ID.

    IF PART_ID NOT = "TERMINAL"
       FIND FIRST PART WITHIN ALL_PART USING PART_ID
            ON ERROR DISPLAY "ERROR IN FIND "
                     PERFORM 200-PARTS-RECORD-ERROR
                     GO TO 150-UPDATE-PARTS-RECORD
       END-FIND
       PERFORM VERIFY-PART-ROUTINE
    ELSE
       PERFORM TERMINAL-ROUTINE.

    DISPLAY "ENTER PART DESCRIPTION - X(50): "
```

```
            WITHOUT ADVANCING.
    ACCEPT PART_DESC.
       .
       .
       .
    MODIFY PART
       ON ERROR DISPLAY "ERROR MODIFYING PARTS_RECORD..."
                PERFORM 200-PARTS-RECORD-ERROR-ROUTINE.

    GO TO 150-UPDATE-PARTS-RECORD.
```

2. In this example, PART_ID in the PART record is modified. PART_ID is a sort key for the CLASS_PART set. This routine will retain the current position in the CLASS_PART set after you modify the record.

```
300-FIX-ID-ROUTINE.

    DISPLAY "ENTER PART ID TO BE CHANGED - X(8): "
            WITHOUT ADVANCING.
    ACCEPT PART_ID.

    FIND  FIRST PART WITHIN CLASS_PART USING PART_ID
       ON ERROR DISPLAY "ERROR IN FIND PARTS_RECORD..."
                PERFORM 200-PARTS-RECORD-ERROR-ROUTINE.

    DISPLAY "ENTER NEW PART ID - X(8): "
             WITHOUT ADVANCING.
    ACCEPT PART_ID.

    MODIFY PART_ID RETAINING CLASS_PART
        ON ERROR DISPLAY "ERROR MODIFYING PARTS_RECORD..."
                 PERFORM 200-PARTS-RECORD-ERROR-ROUTINE.
```

READY

READY — Begins a database transaction, prepares one or more database realms for processing, and places each specified realm in a ready mode.

General Format

READY[realm-name]...

[USAGE-MODE IS { { CONCURRENT | EXCLUSIVE | PROTECTED | BATCH } [{ RETRIEVAL | UPDATE }] | { RETRIEVAL | UPDATE } [{ CONCURRENT | EXCLUSIVE | PROTECTED | BATCH }] }]

[WITH WAIT]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-READY]

realm-name
names a subschema realm.

`stment`
is an imperative statement executed for an on error condition.

`stment2`
is an imperative statement executed for a not on error condition.

General Rules

1. The READY statement begins a database transaction.
2. Execution of the READY statement is successful only when all specified *realm-names* are placed in the ready mode.
3. If you do not specify a *realm-name*, the DBCS (Database Control System) readies all realms in your subschema except those already readied.
4. The USAGE-MODE phrase establishes run-time privileges for this run unit. It also affects other run units running concurrently. It consists of two parts: (a) the allow mode, and (b) the access mode.
5. If the program does not specify a usage mode, PROTECTED RETRIEVAL is the default.
6. The allow mode specifies what you will allow other concurrent run units to do. It consists of the reserved words BATCH, CONCURRENT, EXCLUSIVE, and PROTECTED.
7. The BATCH RETRIEVAL option allows concurrent run units to update the realm. You can retrieve data as though a copy of the database had been made at the time you readied the realm. This eliminates the possibility of deadlocks due to record lock conflicts. Any updates made by concurrent run units will not be available to your transaction.

   If you ready any realm in BATCH, you must ready all realms in BATCH.
8. The BATCH UPDATE option allows you to access or update any data in the realm while preventing concurrent run units from accessing or updating the realm.
9. The CONCURRENT clause permits other run units to update records in the same areas that map to the readied *realm-name*. Record-locking will be done for you by the DBCS to protect the integrity of your currency indicators, keeplists, and uncommitted changes. You will not be able to see any other user's uncommitted changes.
10. The EXCLUSIVE clause prevents concurrent run units from accessing records in the same areas that map to the readied *realm-name*.
11. The PROTECTED clause (the default) prohibits concurrent run units from updating records in the same areas that map to the readied *realm-name*. Concurrent run units will be able to ready the realm for retrieval only.
12. The access mode indicates what your run unit will do. It consists of the reserved words RETRIEVAL and UPDATE.
13. The RETRIEVAL clause (the default) allows your run unit read-only privileges. It prevents your unit from updating records in the readied realms.
14. The UPDATE clause permits your run unit to update records in a readied realm. It allows the run unit to execute any DML statement against the specified realm.
15. The WAIT option tells the DBCS to READY the specified realms as soon as possible. You will get a response only when the realms can be readied or when an error other than a lock conflict occurs.

16. Currency indicators are not affected by the READY statement.

17. If a database exception condition occurs during the execution of a READY statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Notes). This code identifies the condition.

18. Figure 4.1 summarizes the effects of the allow and access options on concurrent run units readying the same *realm-name*. A database exception condition will occur if a concurrent run unit has a *realm-name* readied in a conflicting or incompatible mode (see Technical Notes).

    The First Run Unit column refers to a READY statement that has already been executed in a concurrent run unit. The Second Run Unit column refers to a READY statement being attempted by the current run unit. The intersection indicates whether the attempted READY statement can be executed immediately or if there is a conflict.

    Because Figure 4.1 is symmetrical, the chronology of access (that is, did the First Run Unit ready the realm first) does not matter.

    The locking effects of the BATCH RETRIEVAL usage mode are equivalent to those of the CONCURRENT RETRIEVAL usage mode; however, BATCH RETRIEVAL performs no record locking.

    The locking effects of the BATCH UPDATE usage mode are equivalent to those of the EXCLUSIVE UPDATE usage mode.

    Note also that the two EXCLUSIVE usage modes have identical locking effects.

Figure 4.1. Usage Mode Conflicts

First Run Unit \ Second Run Unit	Concurrent Retrieval	Protected Retrieval	Exclusive Retrieval	Concurrent Update	Protected Update	Exclusive Update
CONCURRENT RETRIEVAL	Yes	Yes	No	Yes	Yes	No
PROTECTED RETRIEVAL	Yes	Yes	No	No	No	No
EXCLUSIVE RETRIEVAL	No	No	No	No	No	No
CONCURRENT UPDATE	Yes	No	No	Yes	No	No
PROTECTED UPDATE	Yes	No	No	No	No	No
EXCLUSIVE UPDATE	No	No	No	No	No	No

Technical Notes

READY statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_ALLREADY	All subschema realms are already readied.
DBM$_READY	*Realm-name* is already readied.
DBM$_AREABUSY	Your usage mode conflicts with another run unit's usage mode. See Figure 4.1.

Additional References

- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements

- Section 4.8.1, on database On Error condition

Example

Depending on the contents of PROCESS-MODE, this statement readies the BUY, MAKE, and PERSONNEL realms in either PROTECTED UPDATE mode or CONCURRENT RETRIEVAL mode (default for USAGE-MODE).

```
IF PROCESS-MODE = "UPDATE"
   READY BUY, MAKE, PERSONNEL
   USAGE-MODE IS PROTECTED UPDATE
   ON ERROR DISPLAY "ERROR READYING ..."
            DISPLAY "IN PROTECTED UPDATE MODE..."
            PERFORM ERROR-ROUTINE
   END-READY
   PERFORM UPDATE-ROUTINE
ELSE
   READY BUY, MAKE, PERSONNEL
   ON ERROR DISPLAY "ERROR READYING ..."
            DISPLAY "IN CONCURRENT RETRIEVAL MODE..."
            PERFORM ERROR-ROUTINE
   END-READY.
```

RECONNECT

RECONNECT — Moves the current database record of the run unit from one set to another (possibly the same) set.

General Format

RECONNECT [record-name]**WITHIN** { {set-name}... | ALL }

[RETAINING [[{ REALM | RECORD | { SET [set-name]... | {set-name}... } }]] CURRENCY]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-RECONNECT]

record-name
names a subschema record type.

set-name
names a subschema set type.

stment
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

Syntax Rules

1. The record type of *record-name* must be a member record type of the set type for each *set-name*.

2. *Set-name* cannot be specified more than once in the WITHIN clause.

3. *Set-name* cannot be specified more than once in the RETAINING clause.

General Rules

1. RECONNECT uses the current record of the run unit.

2. The current run unit record must be in a realm in ready update mode and all records required by the Database Control System (DBCS) to execute the RECONNECT statement must be in realms in available mode.

3. Use *record-name* to check that the record type of the current record of the run unit is identical to the record type specified by *record-name*.

4. For each *set-name* you specify in the WITHIN clause:

   a. The DBCS disconnects the current record of the run unit from the set in which it is currently a member and inserts this record in the current set of the *set-name* set type.

   b. The position where the DBCS inserts the record into the set is determined by the set-ordering criteria defined in the schema for *set-name*.

5. If the program specifies the ALL option:

   a. The DBCS considers only those set types that satisfy these requirements:

      - The set type is in your subschema.

      - The current run-unit record type is defined in the schema as an OPTIONAL or MANDATORY member record type of the set type.

      - The current run-unit record is presently a member of a set of the set type.

   b. For each selected set type:

      - The DBCS disconnects the current record of the run unit from the set in which it is currently a member and inserts this record in the current set of the selected set type.

      - The position where the DBCS inserts the record into the set is determined by the set-ordering criteria defined in the schema for *set-name*.

6. Unless otherwise specified by the RETAINING clause (see Section 4.10: RETAINING Clause), the DBCS updates the set type currency indicators for the reconnected sets to point to the connected record.

7. If a database exception condition occurs during the execution of a RECONNECT statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Notes). This code identifies the condition.

Technical Notes

RECONNECT statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_CRUN_NULL	The currency indicator for the run unit is null.

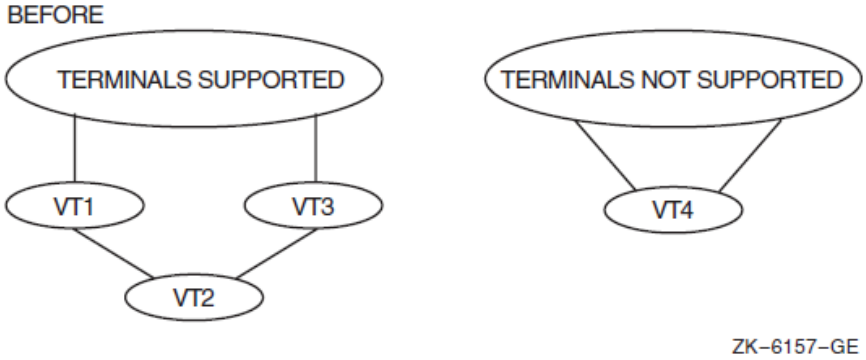
DBM$_CRUN_POS	The currency indicator for the run unit points to a vacated position in the database.
DBM$_WRONGRTYP	The record type of record-name is not identical to the current record type.
DBM$_CSTYP_NULL	There is no current of set type for the set specified in the TO set-name phrase. This occurs only if the set is not a singular set.
DBM$_DUPNOTALL	A sort key data item in the record to be reconnected is the same as the sort key of a record already in the set.
DBM$_FIXED	The program attempted to reconnect a FIXED member record from one set type to another set of a given set type.
DBM$_NOT_UPDATE	The realm is not in ready update usage mode.
DBM$_NOT_MBR	You attempted to reconnect a record to a set in which it is not a member.
DBM$_CHKMEMBER	The Oracle CODASYL DBMS CHECK (member) condition was evaluated to be false. The database has not been changed.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- Section 5.14.1, on RETAINING clause
- USE statement

Example

This example shows how to change a record's set membership (terminal classifications) when the record (VT3) is a MANDATORY member. (OPTIONAL members must be DISCONNECTED from the old set occurrence and CONNECTED to the new set occurrence).

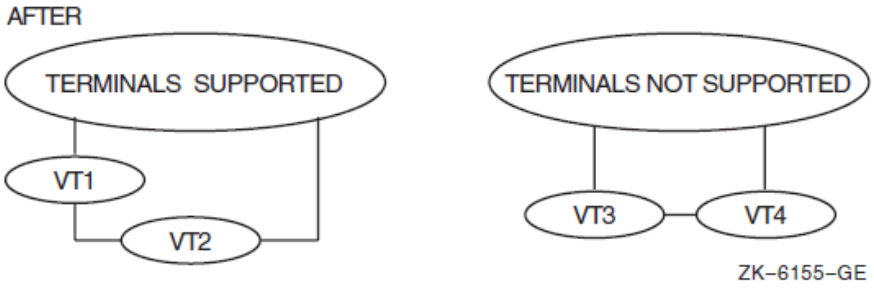



```
MOVE "TERMINALS NOT SUP" TO CLASS_DESC.
FIND FIRST CLASS-REC WITHIN ALL_CLASS
      USING CLASS_DESC.

MOVE "VT3" TO PART_DESC.
FIND FIRST PART WITHIN ALL_PARTS_ACTIVE.
```

```
      USING PART_DESC
      RETAINING CLASS_PART.

    RECONNECT PART WITHIN CLASS_PART.
```



ROLLBACK

ROLLBACK — Ends your database transaction, nullifies all database changes made by this run unit since its last quiet point, and establishes a new quiet point for this run unit.

General Format

ROLLBACK [STREAM]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-ROLLBACK]

stment
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

Syntax Rules

The STREAM clause cannot be specified if the subschema entry (DB) does not name a stream.

General Rules

1. The ROLLBACK statement ends your database transaction.
2. All keeplists are emptied.
3. If you do not use the STREAM clause, the ROLLBACK statement does the following:
   - Nullifies all database changes made by the run unit since the last quiet point
   - Terminates the ready mode of all ready realms for the run unit
   - Establishes a new quiet point for the run unit
4. If you use the STREAM clause, the ROLLBACK statement does the following:

- Nullifies all database changes made by this stream since the last quiet point
- Terminates the ready mode of all ready realms for this stream
- Establishes a new quiet point for this stream

5. All currency indicators are set to null.
6. All realm and record locks are released. These records and realms are now available to concurrent run units.
7. To begin another transaction, the program must execute another READY statement after it executes the ROLLBACK statement.
8. If the run unit abnormally terminates, the DBCS executes an implicit ROLLBACK statement.
9. If a database exception condition occurs during the execution of a ROLLBACK statement, the DBCS places a database exception condition code in the special register DB-CONDITION. This code identifies the condition.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- USE statement (USE FOR DB-EXCEPTION)
- Section 5.2.4, on subschema description (DB)

Example

This ROLLBACK example illustrates one way to end a database transaction and undo the changes made to the database during the transaction.

```
010-BEGIN-UPDATE-TRANSACTION.

* This transaction begins with the first READY statement in
* the run unit, continues through a series of DML database
* access statements, and ends with a COMMIT or ROLLBACK.
    .
    .
    .

100-END-UPDATE-TRANSACTION.

    DISPLAY "DO YOU WANT TO COMMIT THIS TRANSACTION ? ".
    ACCEPT ANSWER.
    IF ANSWER = "YES"
       COMMIT ...
     ELSE
       ROLLBACK
            ON ERROR
               IF DB-CONDITION = DBM-NOT-BOUND
```

```
            DISPLAY " Database not bound"
         ELSE
            CALL "DBM$SIGNAL".
         END-IF
      END-ROLLBACK.
```

STORE

STORE — Stores a new record in the database, establishes the record as an owner of an empty set of each set type for which the record is an owner record type, and connects the record as a member to the current set of each set type for which the record is an AUTOMATIC member record type.

General Format

STORE record-name [[NEXT TO] DBKEY] [WITHIN {realm-name}...]

[RETAINING [[{ REALM | RECORD | { SET [set-name]... | {set-name}... } }]] CURRENCY]

[ON ERROR stment]

[NOT ON ERROR stment2]

[END-STORE]

record-name
names a subschema record type.

realm-name
names a subschema realm.

set-name
names a subschema set type.

stment
is an imperative statement executed for an on error condition.

stment2
is an imperative statement executed for a not on error condition.

Syntax Rules

1. *Realm-name* cannot be specified more than once in the same STORE statement.
2. *Set-name* cannot be specified more than once in the same STORE statement.

General Rules

1. The STORE statement references the *record-name* in the user work area. You must move the data to be in *record-name* to the user work area before executing a STORE statement.
2. If you specify the WITHIN option, the Database Control System (DBCS) stores a new record occurrence in one of the realms from the list of *realm-name*s.

   If you do not specify the WITHIN option, the DBCS stores a new record occurrence in one valid subschema realm for that record type.

3. If you specify the DBKEY option, the target area is determined by page size. If the page specified by the DB-KEY special register has space available, that page is the target area. Otherwise, the DBCS chooses the next page that has available space.

4. After a successful STORE operation, the DB-KEY special register contains the database key for the record.

5. The STORE statement stores a record occurrence of the *record-name* record type from your user work area to a single target area.

6. If the DBCS can store *record-name* in more than one realm or area, the selected realm or area is unspecified.

7. The successful STORE statement directs the DBCS to store these items in the target area:

   - Each *record-name* data item defined in the subschema.

   - The default value for any *record-name* data items defined in the schema but not defined (omitted) in the subschema. The schema DEFAULT clause defines the values for these omitted data items.

8. The DBCS establishes the newly stored record as the owner record of an empty set for each set type for which *record-name* is defined as the owner record type.

9. The DBCS connects the newly stored record as a member record of the current set of each set type for which *record-name* is defined as an AUTOMATIC member record type. The set criteria defined in the schema for that set type determine the position where the DBCS inserts *record-name*.

10. Unless otherwise specified by the RETAINING clause (see Section 4.10: RETAINING Clause), these currency indicators point to the stored record:

    - Run unit

    - Realm

    - Record type

    - Set type for each set type the record owns, and for each set type of which it is an AUTOMATIC member

11. The contents of the user work area do not change after the successful or unsuccessful execution of a STORE statement.

12. If a database exception condition occurs during the processing of a STORE statement, the DBCS places a database exception condition code in the special register DB-CONDITION (see Technical Notes). This code identifies the condition.

Technical Notes

STORE statement execution can result in these DB-CONDITION database exception condition codes:

DBM$_NODEFVAL	There is no schema DEFAULT clause for an omitted data item in *record-name*.
DBM$_CHKITEM	A *record-name* item contains an invalid value as determined by a schema CHECK clause.

DBM$_CHKMEMBER	A *record-name* item contains an invalid value as determined by a schema CHECK clause.
DBM$_CHKRECORD	A *record-name* item contains an invalid value as determined by a schema CHECK clause.
DBM$_CSTYP_NULL	The set currency indicator for an AUTOMATIC, nonsingular set type in which *record-name* is a member is null.
DBM$_CONVERR	A data conversion error occurred in the STORE operation.
DBM$_ILLNCHAR	Invalid character found in a numeric field.
DBM$_NONDIGIT	Nonnumeric character found in a numeric field.
DBM$_OVERFLOW	A data overflow error occurred in the STORE operation.
DBM$_TRUNCATION	A data truncation error occurred in the STORE operation.
DBM$_UNDERFLOW	A data underflow error occurred in the STORE operation.
DBM$_SETSELECT	You cannot store this record using this subschema. Either add the specified set type to your subschema or use a different subschema.

Additional References

- Section 2.2, on reserved words (database special registers)
- *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/], on scope of statements
- Section 4.8.1, on database On Error condition
- Section 5.14.1, on RETAINING clause
- USE statement

Example

```
010-ADD-NEW-CLASS-RECORDS.

    DISPLAY "ENTER CLASS CODE".
    ACCEPT  CLASS_CODE.
    DISPLAY "ENTER CLASS DESCRIPTION".
    ACCEPT  CLASS_DESC.
    DISPLAY "ENTER CLASS STATUS".
    ACCEPT  CLASS_STATUS.

    STORE CLASS_REC WITHIN MAKE
          ON ERROR DISPLAY "ERROR STORING CLASS..."
                   PERFORM 200-STORE-CLASS-ERROR
          END-STORE.
```

USE

USE — Specifies Declaratives procedures to handle file input/output errors and database exception conditions. It can also specify procedures to be executed before the program processes a specific report group. These procedures supplement the procedures in the COBOL Run-Time System and RMS. USE is part of the COBOL ANSI standard.

General Format

Formats 1 and 2

Refer to the *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/] for Formats 1 and 2. Only Format 3 is applicable to DBMS DML.

Format 3

USE [GLOBAL] **FOR DB-EXCEPTION**

[ON{ {DBM$_exception-condition}... | OTHER }] **.**

DBM$_exception-condition
is a symbolic constant name beginning with the characters DBM$_. It identifies an Oracle CODASYL DBMS exception condition. Refer to the Oracle CODASYL DBMS documentation set for information on DML error and warning messages.

Syntax Rules

All Formats

1. A USE statement can be used only in a sentence immediately after a section header in the Procedure Division Declaratives area. It must be the only statement in the sentence. The rest of the section can contain zero, one, or more paragraphs to define the particular USE procedures.

2. The USE statement itself does not execute. It defines the conditions that cause execution of the associated USE procedure.

3. Refer to *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/] for information on USE formats 1 and 2.

Format 3

1. The *DBM$_exception-condition* argument must begin with these five characters: DBM$_.

2. If the phrase USE [GLOBAL] FOR DB-EXCEPTION. occurs, it is the only USE procedure allowed in the program.

3. Multiple occurrences of the USE statement can exist within a program only if the ON phrase is specified. *DBM$_exception-condition* in each USE statement must be unique. Multiple USE statements must not cause the simultaneous request for execution of more than one USE procedure. The OTHER phrase may be specified only once in a program.

4. If a USE FOR DB-EXCEPTION statement does not specify the ON phrase, it must be the only occurrence of Format 3 of the USE statement in the program.

General Rules

All Formats

1. At run time, two special precedence rules apply for the selection of a USE procedure when a program is contained within another program. In applying these rules, only the first qualifying USE procedure is selected for execution. The order of precedence for the selection of a USE procedure is as follows:

- First, select the USE procedure within the program containing the statement that caused the qualifying condition.
- If a USE procedure is not found in the program using the previous rule, the Run-Time System searches all programs directly or indirectly containing that program for a USE GLOBAL ... Declaratives procedure. This search continues until the Run-Time System either: (a) finds an applicable USE GLOBAL Declaratives procedure, or (b) finds the outermost containing program, if there is no applicable USE GLOBAL Declaratives. Either condition terminates the search.

2. A USE procedure cannot refer to a non-Declaratives procedure. However, only the PERFORM statement can transfer execution control from:
   - A Declaratives procedure to another Declaratives procedure
   - A non-Declaratives procedure to a Declaratives procedure
3. After a USE procedure executes, control returns to the next executable statement in the invoking routine, if one is defined. Otherwise, control transfers according to the rules for Explicit and Implicit Transfers of Control.
4. A program must not execute a statement in a USE procedure that would cause execution of a USE procedure that had been previously executed and had not yet returned control to the routine that invoked it.

Format 3

1. Prior to the execution of a database USE procedure, the DBCS places appropriate values in the DB-CONDITION, DB-CURRENT-RECORD-NAME, and DB-CURRENT-RECORD-ID special registers.
2. A database USE procedure executes automatically:
   - After standard database exception condition processing ends
   - When an on error condition or an at end condition results from a COBOL DML (data manipulation language) statement that has no applicable ON ERROR or AT END clause
3. Use of a DML verb in a USE FOR DB-EXCEPTION is not supported.
4. If there is an applicable USE FOR DB-EXCEPTION procedure, it executes whenever a database exception condition occurs. However, it does not execute if: (a) the condition is at end and there is an AT END phrase or (b) the condition is any database exception and there is an ON ERROR phrase.
5. If the DBCS (Database Control System) detects more than one database exception condition during the execution of a COBOL DML statement, the database USE procedure to which control transfers is determined by the contents of DB-CONDITION.
6. One COBOL DML error cannot cause more than one USE FOR DB-EXCEPTION procedure to execute.
7. A database USE procedure executes for a COBOL DML statement's exception condition: (a) if the OTHER phrase is the only phrase specified in the program, and (b) if these conditions are also true:
   - The COBOL DML statement has no ON ERROR phrase
   - The COBOL DML statement has no AT END phrase

- The program does not specify an ON phrase for that exception condition

After the USE procedure executes, control returns to the next executable statement in the invoking routine, if one is defined. Otherwise, control transfers according to the rules for Explicit and Implicit Transfers of Control.

8. If the ON *DBM$_exception-condition* phrase is specified and a COBOL DML statement results in a database exception condition, execution continues as follows:

   - If the value of DB-CONDITION is equal to a *DBM$_exception-condition*, the procedures associated with that *DBM$_exception-condition* execute.
   - If the value of DB-CONDITION is not equal to any *DBM$_exception-condition*, the procedures associated with the OTHER phrase execute.

9. If a database exception condition occurs and there is no applicable USE FOR DB-EXCEPTION procedure, uncommitted transactions roll back and the image terminates abnormally.

Additional Reference

Section 4.3, Scope of Names

Example

(The Technical Notes following this example explain execution of the USE procedures shown).

```
PROCEDURE DIVISION.
DECLARATIVES.
200-DATABASE-EXCEPTIONS SECTION.
    USE FOR DB-EXCEPTION ON OTHER.
201-PROCEDURE.
    DISPLAY "DATABASE EXCEPTION CONDITION".
    PERFORM 250-DISPLAY-MNEMONIC.
210-DATABASE-EXCEPTIONS SECTION.
    USE FOR DB-EXCEPTION ON DBM$_NOTIP.
211-PROCEDURE.
    DISPLAY "DATABASE EXCEPTION CONDITION ON READY STATEMENT"
    PERFORM 250-DISPLAY-MNEMONIC.
250-DISPLAY-MNEMONIC.

*
*   DBM$SIGNAL displays a diagnostic message based on the
*   status code in DB-CONDITION.
*
*
    CALL "DBM$SIGNAL".

    STOP RUN.

END DECLARATIVES.
```

Technical Notes

- If this program has an exception condition when executing a READY statement, or if the program does not ready the database before executing a DML statement, 210-DATABASE-EXCEPTIONS executes.

- If any other database exception condition occurs during program execution, 200-DATABASE-EXCEPTIONS executes.

4.10. RETAINING Clause

The RETAINING clause specifies which currency indicators are not updated during the execution of the COMMIT, CONNECT, FETCH, FIND, MODIFY, RECONNECT, and STORE statements.

General Format

[RETAINING [[{ REALM | RECORD | { SET [set-name]... | {set-name}... } }]] CURRENCY]

`set-name`
is a set name which is available in the program's subschema.

You can prevent currency indicator updates for REALM, RECORD, or SET. Specifying REALM retains all realm currency indicators; specifying RECORD retains all record currency indicators.

If SET is used without the *set-name* list option, the Database Control System (DBCS) retains the currency indicators for all sets in the subschema. If you specify the *set-name* list option, only the currency indicators for the specified sets are retained. The same *set-name* cannot be listed more than once in a given *set-name* list.

If you specify the RETAINING clause and do not select an option, the DBCS retains all REALM, RECORD, and SET currency indicators.

Chapter 5. Database Programming with VSI COBOL for OpenVMS

With VSI COBOL for OpenVMS database programming you can access data without designing separate files for specific applications. You accomplish this with the database management system (Oracle CODASYL DBMS) and the COBOL data manipulation language (DML). This chapter is a resource for information on:

- VSI COBOL for OpenVMS database program development
- VSI COBOL for OpenVMS database concepts
- VSI COBOL for OpenVMS programming tips and techniques
- Debugging and testing VSI COBOL for OpenVMS database programs

Database programmers and readers unfamiliar with Oracle CODASYL DBMS concepts and definitions should run the online self-paced demonstration package (see Section 5.1) as a prerequisite to this chapter. The demonstration package lets you test Oracle CODASYL DBMS features and concepts as you learn them. Additional useful information can be found in:

- *Introduction to Oracle CODASYL DBMS*
- *Oracle CODASYL DBMS Database Administration Reference Manual*

5.1. The Self-Paced Demonstration Package

To help you learn how to use a database, Oracle has provided you with a database called PARTS. PARTS is an online self-paced demonstration database configured to show some of the features of Oracle CODASYL DBMS. You create the PARTS database as part of the demonstration package. Examples in this chapter refer to either the PARTSS1 or PARTSS3 subschema in the PARTS database. A complete listing of the PARTS schema, including the PARTSS1 and PARTSS3 subschemas, can be found in the Oracle CODASYL DBMS documentation on data manipulation.

Before beginning the demonstration, you should do the following:

1. Create your own node in Oracle CDD/Repository using the Dictionary Management Utility (DMU). (Refer to the Oracle CDD/Repository documentation for more information).

   ```
   $ RUN SYS$SYSTEM:DMU RET
   DMU> CREATE nodename RET
   DMU> SHOW DEFAULT RET
       defaultname
   DMU> EXIT RET
   $
   ```

   where:

nodename	names the new node in the CDD to contain your personal PARTS database.

defaultname	is your CDD default.

For example:

```
$ RUN SYS$SYSTEM:DMU RET
DMU> CREATE DEMONODE RET
DMU> SHOW DEFAULT RET
    CDD$TOP
DMU> EXIT RET
$
```

2. Define CDD$DEFAULT using the defaultname, a period, and the nodename from step 1. For example:

   ```
   $ DEFINE CDD$DEFAULT "CDD$TOP.DEMONODE"
   ```

   where `CDD$TOP` is a defaultname, `.` is a separator period and `DEMONODE` is a nodename.

As of Version 7.0, DBMS ships a kit that allows you to choose either a standard or multi-version DBMS environment.

The multi-version environment is the same as the standard environment, except that it allows multiple versions of DBMS to exist concurrently on the same system. This is implemented by adding the major and minor version number to the image file names, the command procedures that exist in common locations, and the [.DBM] subdirectory located in SYS$COMMON:[SYSTEST].

To run the demonstration package after a standard installation, (of any version of DBMS) issue the following command:

```
$ @SYS$COMMON:[SYSTEST.DBM]DBMDEMO RET
```

To run the demonstration package after a multi-version installation, issue the following command:

```
$ @SYS$COMMON:[SYSTEST.DBMnn]DBMDEMO RET
```

where *nn* is the DBMS major and minor version.

For example, to run the demonstration package on OpenVMS I64 with DBMS Version 7.2, issue the following command:

```
$ @SYS$COMMON:[SYSTEST.DBM72]DBMDEMO RET
```

You must run the entire demonstration to create and load the PARTS database. If you have already created the PARTS database but are unsure of or have changed its contents, you can reload it by running option 11 of the self-paced demonstration package. The demonstration package creates the NEW.ROO database instance. If you have any problems with the demonstration package, see your system manager or database administrator.

5.2. Concepts and Definitions

Some of the important concepts in database programming are described in the definitions of *databases*, *schemas*, and *streams*.

5.2.1. Database

A database is a collection of your organization's data gathered into a number of logically related units. The database administrator (DBA) and representatives from user departments decide on the organization's informational needs. After these individuals agree on the contents of the database, the DBA assumes responsibility for designing, creating, and maintaining the database.

5.2.2. Schema

The schema is a program written by the DBA using DDL statements. It describes the logical structure of the database, defining all record types, set types, areas, and data items in the database. The DBA writes the schema independently of any application run unit. Only one schema can exist for a database. For a more detailed description of the schema DDL, refer to the Oracle CODASYL DBMS documentation on database administration and design.

5.2.3. Storage Schema

The storage schema describes the physical structure of the database. It is written by the DBA using data storage description language (DSDL) statements. For a complete description of the storage schema, refer to the Oracle CODASYL DBMS documentation on database administration and design.

5.2.4. Subschema

The subschema is a subset of the schema; it is your run unit's view of the database. The DBA uses the subschema DDL to write a subschema, defining only those areas, set types, record types, and data items needed by one or more run units. You specify a subschema to be used by your run unit with the DB statement. A subschema contains data description entries like the record description entries you use for file processing. However, subschema data description entries are not compatible with COBOL data description entries; the VSI COBOL for OpenVMS compiler must translate them. The translated entries are made available to the COBOL program at compile time. By using the /MAP compiler qualifier, you obtain a database map showing the translated entries as part of your program listing.

Many subschemas can exist for a database. For further information on writing a subschema, refer to the *Oracle CODASYL DBMS Database Administration Reference Manual*.

5.2.5. Stream

A stream is an independent access channel between a run unit and a database. A stream has its own keeplists, locks, and currency indicators. You specify a stream to be used by your run unit with the DB statement. Streams let you do the following:

- Access multiple subschemas within the same database
- Access multiple databases

Because streams can lock against one another, it is possible to deadlock within a single process.

In VSI COBOL for OpenVMS, you can only specify one stream per separately compiled program. To access multiple subschemas within the same database or multiple databases, you must use multiple separately compiled programs and execute calls between the programs. For example, to gain multiple access to the databases OLD.ROO and NEW.ROO, you could set up a run unit as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  MULTI-STREAM-1.
DATA DIVISION.
SUB-SCHEMA SECTION.
```

```
DB PARTS1 WITHIN PARTS FOR "NEW.ROO" THRU STREAM-1.
    .
    .
    .
    CALL MULTI-STREAM-2
    .
    .
    .

END PROGRAM MULTI-STREAM-1.
IDENTIFICATION DIVISION.
PROGRAM-ID.  MULTI-STREAM-2.
DATA DIVISION.
SUB-SCHEMA SECTION.
DB DEFAULT_SUBSCHEMA WITHIN PARTS FOR "NEW.ROO" THRU STREAM-2.
    .
    .
    .
    CALL MULTI-STREAM-3.
    EXIT PROGRAM.
IDENTIFICATION DIVISION.
PROGRAM-ID.  MULTI-STREAM-3.
DATA DIVISION.
SUB-SCHEMA SECTION.
DB OLDPARTS1 WITHIN OLDPARTS FOR "OLD.ROO" THRU "STREAM-3".
    .
    .
    .
    EXIT PROGRAM.
```

In this run unit, the main program (MULTI-STREAM-1) accesses the database NEW.ROO through STREAM-1 and performs a call to a subprogram. The subprogram (MULTI-STREAM-2) accesses another subschema to the database NEW.ROO through STREAM-2 and calls another subprogram. This subprogram (MULTI-STREAM-3) accesses a second database (OLD.ROO) through STREAM-3.

STREAM-1, STREAM-2, and STREAM-3 are stream names. Stream names assign a character string name to the database/subschema combination you specify in your DB statement. For more information, refer to the *VSI COBOL Reference Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-reference-guide/] and the Oracle CODASYL DBMS documentation.

5.3. Using Oracle CDD/Repository

Oracle CODASYL schemas, storage schemas, and subschemas are stored in Oracle CDD/Repository. Oracle CDD/Repository separates data descriptions from actual data values that reside in VMS files. (For more information, refer to the Oracle CODASYL DBMS documentation on Common Data Dictionary Utilities and the Oracle CDD/Repository documentation.) Because of this separation, VSI COBOL for OpenVMS DML programs can be written independently of data. In addition, several subschemas can describe the same data according to their particular needs. This eliminates the need for redundant data and ensures data integrity.

At compile time, the COBOL DB statement, in effect, references Oracle CDD/Repository to obtain the data descriptions of a specific subschema. It is not until run time that the COBOL program has access to the database data values.

5.4. Database Records

A database record, like a record in a file, is a named collection of elementary database data items. Records appear in the database as record occurrences. Oracle CODASYL DBMS records are linked into sets.

In VSI COBOL for OpenVMS database applications, you do not describe database records in the COBOL program. Rather, you must use the DB statement to extract and translate subschema record definitions into your COBOL program as COBOL record definitions.

Each record description entry defined by the DBA in the schema describes one record type (see Section 5.7). For example, in Figure 5.7, PART is one record type and SUPPLY is another record type. Any number of records can be stored in a database.

In Oracle CODASYL DBMS, records are also called record occurrences. Figure 5.6 shows one occurrence of PART record type and two occurrences of SUPPLY record type.

The subschema describes records that you can access in your program. Note that subschema record descriptions might define only a portion of a schema record. For example, if a schema record description is 200 characters long, a corresponding subschema record description could be less than 200 characters long and use different data types.

Individual database records are locked by the DBCS as they are retrieved by the run unit, and the degree of locking depends on the specific DML command used. For more information, see Section 6.1.1.

5.5. Database Data Item

A database data item is the smallest unit of named data. Data items occur in the database as data values. These values can be character strings or any of several numeric data types.

5.6. Database Key

A database key (dbkey) identifies a record in the database. The value of the database key is the storage address of the database record. You can use this key to refer to the record pointed to by a currency indicator or an entry in a keeplist. For example, KEEP, FIND ALL, and FREE statements store and release these values from a keeplist you define in the subschema section.

5.7. Record Types

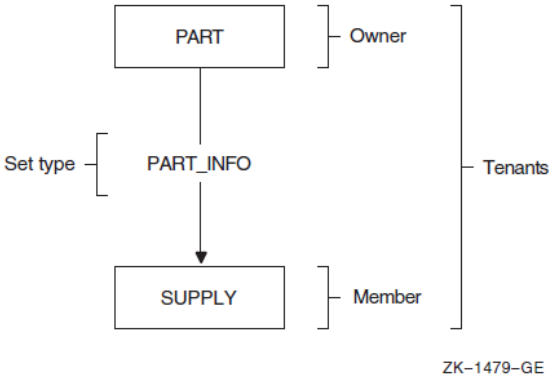
Records are grouped according to common features into record types. The database administrator (DBA) describes record types in the schema; record occurrences exist in the database. For example, a record that contains a specific part name, weight, and cost is a record occurrence. The PART record type, describing the structure of all occurrences of part records, would be defined in the schema. The unqualified term record implies record occurrence.

5.8. Set Types

A set type is a named relationship between two or more record types. The major characteristic of a set type is a relationship that relates one or more member records to an owner record. The owner and members of a set are called tenants of the set. For example, the PART record type could own a SUPPLIER record type in the set PART_INFO.

As with records, the DBA describes set types in the schema; set occurrences exist in the database. The unqualified term set implies set occurrence. A set occurrence is the actual data in the set, not its definition, which is the set type. Figure 5.1 illustrates a set relationship using a Bachman diagram.

Figure 5.1. Bachman Diagram



A Bachman diagram shows how member records are linked with owner records by arrows that point toward the members. It is a graphic representation of the set relationships between owner and member records used to analyze and document a database design. This simple format can be extended to describe many complex set relationships. The Oracle CODASYL DBMS documentation on data manipulation contains a complete Bachman diagram of the PARTS database.

Most of the examples in this chapter use the set types in the PARTSS1 and PARTSS3 subschemas (see the subschema compiler listings in Section 7.3, and the Bachman diagrams in this chapter, Figure 5.2 and Figure 5.3). Figure 5.4 and Figure 5.5 contain three PART records, two VENDOR records, and six SUPPLY records. The SUPPLY records show suppliers' lag times. Lag time starts when an item is ordered and ends when the item is received.

The examples assume the records are in the following order:

1. PART record type: LABEL, CASSETTE, TAPE
2. SUPPLY record type: 4-DAYS, 2-DAYS, 1-MONTH, 1-WEEK, 2-WEEKS, 5-DAYS
3. VENDOR record type: MUSICO INC., SOUND-OFF CO.

Note

All occurrence diagrams display member records within a set in counterclockwise order.

Figure 5.2. Partial Bachman Diagram of the PARTSS1 Subschema

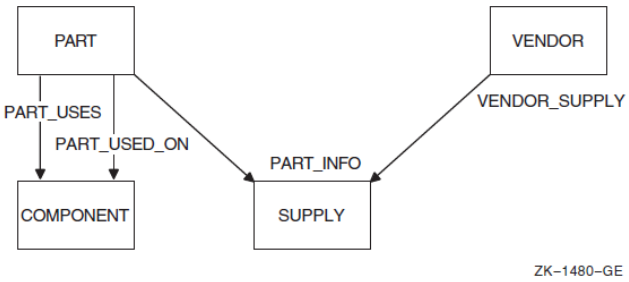


Figure 5.3. Bachman Diagram of the PARTSS3 Subschema

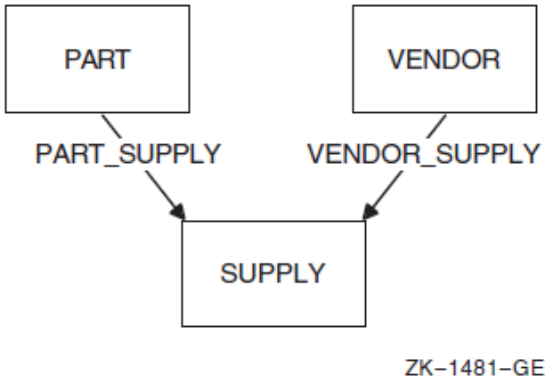


Figure 5.4. Sample Occurrence Diagram 1

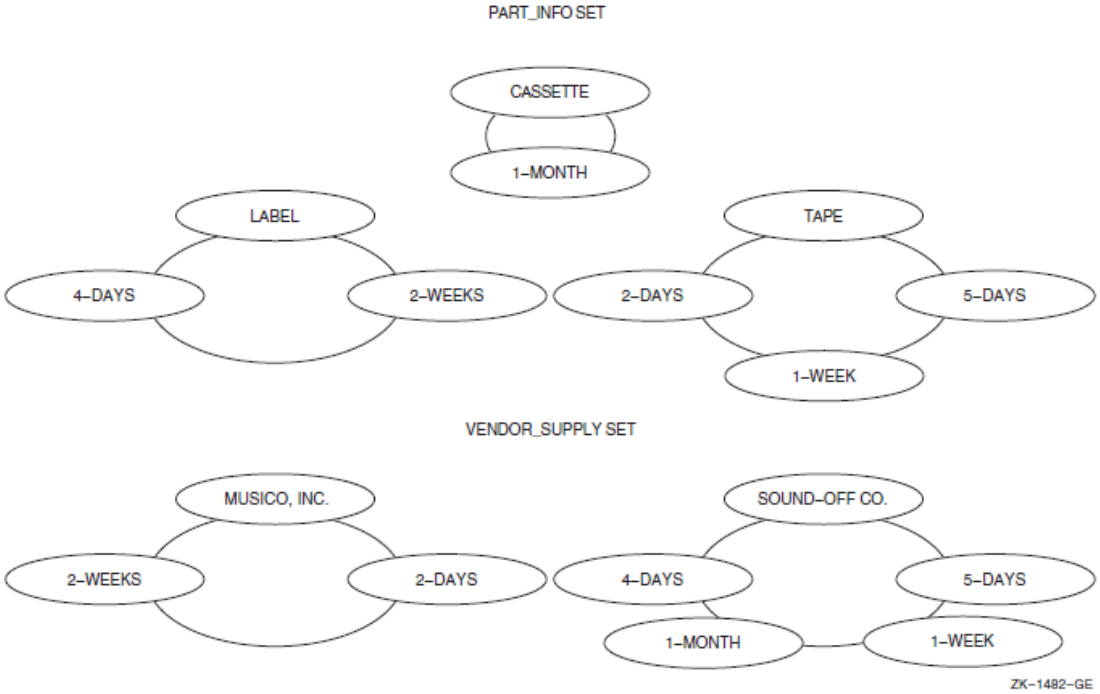
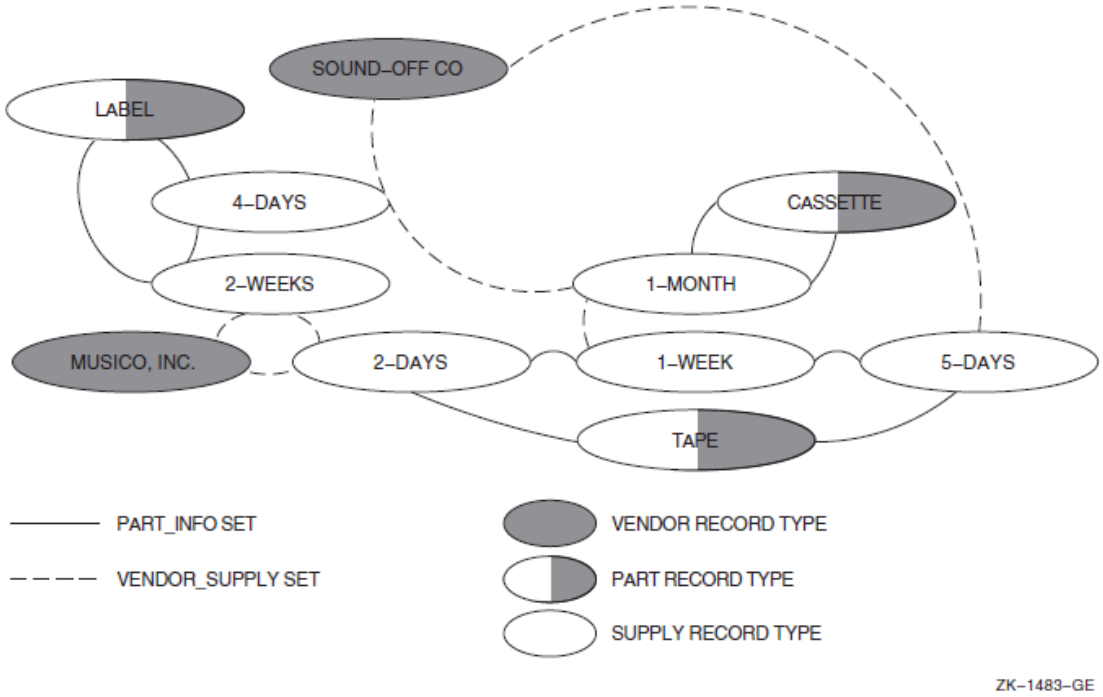


Figure 5.5. Sample Occurrence Diagram 2

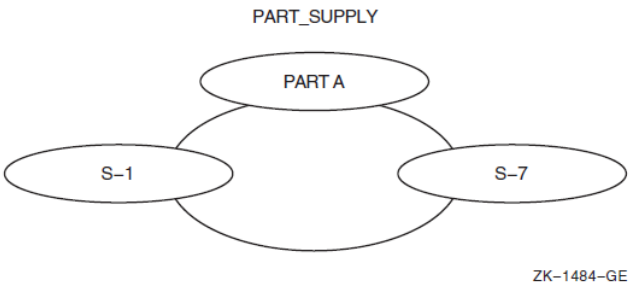


5.9. Sets

Sets are the basic structural units of a database. A set occurrence has one owner record occurrence and zero, one, or several member record occurrences. Figure 5.6 shows one occurrence of PART_SUPPLY set where PART A owner record occurrence owns two SUPPLY member record occurrences.

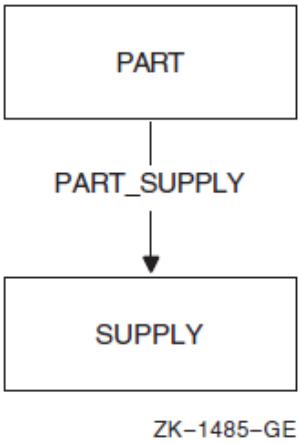
Set types establish a logical relationship between two or more types of records. A subschema usually includes one or more set types. Each set type has one record type that participates as the owner record and one or more record types that participate as members. These owner and member records are grouped into set occurrences.

Figure 5.6. One Occurrence of Set PART_SUPPLY



The DBA can specify a set type where each PART record occurrence can own SUPPLY record occurrences. Figure 5.7 is a Bachman diagram that shows the relationship between PART record types and SUPPLY record types. Bachman diagrams give you a picture of the schema or a portion of the schema. Each record type is enclosed in a box. Each set type is represented by an arrow pointing from the owner record type to the member record type or types. Thus, in Figure 5.7, PART is the owner record type of the PART_SUPPLY set type, and SUPPLY is the member record type.

Figure 5.7. Set Relationship



You can have many set relationships in a subschema. Figure 5.8 shows a set relationship where vendor records are also owners of supply records. You would use this relationship when many parts are supplied by one vendor, and many vendors supply one part. For example, Figure 5.9 shows a gasket supplied by three vendors. The supply records show the minimum quantity each vendor is willing to ship.

Figure 5.8. Set Relationships

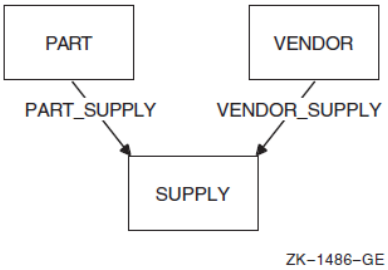
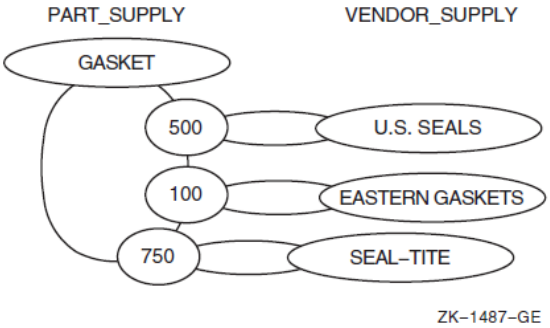


Figure 5.9. Occurrence Diagram of a Relationship Between Two Set Types



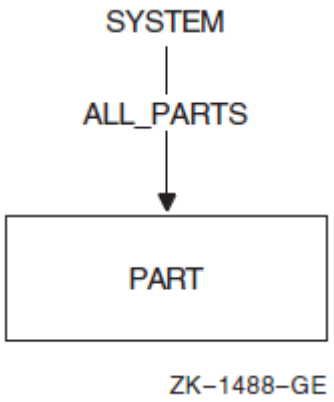
5.9.1. Simple Set Relationships

A simple set relationship contains one owner record type and one or more member record types. Simple relationships are used to represent a basic one-to-many relationship where one owner record occurrence owns zero, one, or several member record occurrences. Simple relationships are created with a single set type. There are three kinds of sets in simple relationships: system-owned sets, simple sets, and forked sets.

5.9.1.1. System-Owned Sets

By definition, a set contains one owner record and may contain zero or more member records. Sets owned by the system, however, have only one occurrence in the database and are called system-owned sets. System-owned sets are used as entry points into the database. You cannot access the owner of a system-owned set (the system), but you can access its member records. System-owned sets are also called singular sets. Figure 5.10 is an example of a system-owned set type.

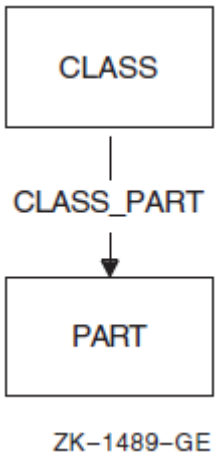
Figure 5.10. Bachman Diagram of a System-Owned Set Type



5.9.1.2. Simple Sets

In simple sets, each set contains only one type of member record. Figure 5.11 is a Bachman diagram of a simple set type where similar parts are grouped by class code. For example, plastic parts could be member records owned by a class record with a class code PL.

Figure 5.11. Bachman Diagram of a Simple Set Type



Example 5.1 prints a listing of all parts with a class code of PL.

Example 5.1. Printing a Listing of a Simple Set

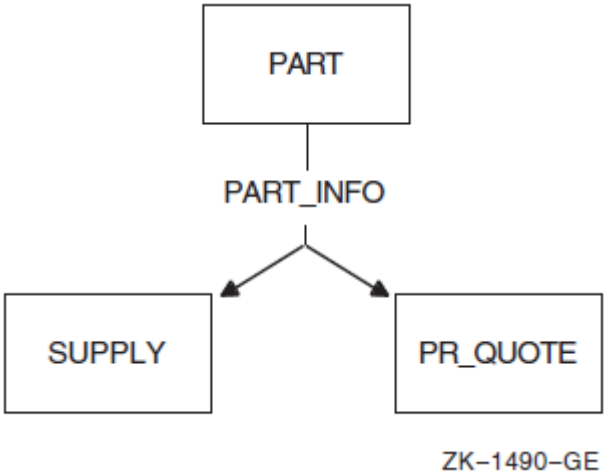
```
PROCEDURE DIVISION.
    .
    .
    .
100-GET-PLASTICS-CLASS.
    MOVE "PL" TO CLASS_CODE
```

```
    FIND FIRST CLASS USING CLASS_CODE.
200-GET-PLASTICS-PARTS.
    FETCH NEXT PART WITHIN CLASS_PART
      AT END GO TO 900-DONE-PLASTIC-PARTS.
   ****************************************************
   *   Plastic parts print routine.
   ****************************************************
      GO TO 200-GET-PLASTICS-PARTS.
```

5.9.1.3. Forked Sets

A forked set has one owner record type and members of two or more different member record types. In most forked sets, the member record types have common data characteristics. One such example is the set type PART_INFO in Figure 5.12, where member record types SUPPLY and PR_QUOTE both contain information about parts.

Figure 5.12. Bachman Diagram of a Forked Set Type



One advantage of a forked set type is the ability to connect many different record types to one set type. Another advantage is that owner records need only one set of pointers to access more than one member record type. Example 5.2 uses the forked set type shown in Figure 5.12 and the forked set occurrence in Figure 5.13 to perform a part analysis.

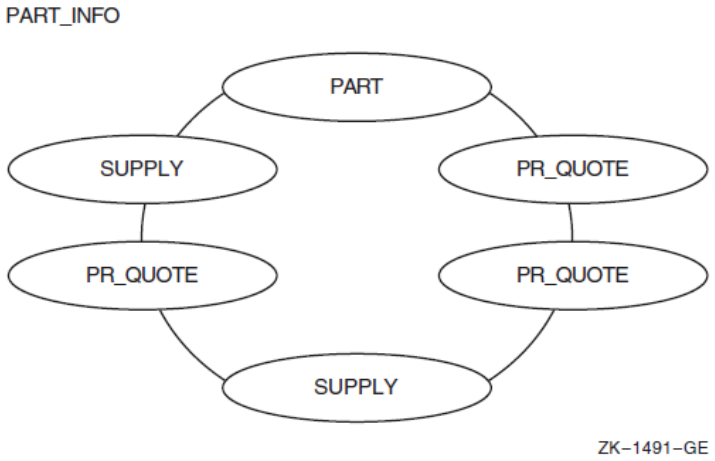
Example 5.2. Using Forked Sets

```
PROCEDURE DIVISION.
    .
    .
    .
100-GET-PART.
    DISPLAY "TYPE PART ID".
    ACCEPT PART_ID.
    IF PART_ID = "DONE"
       GO TO 900-DONE-PART-INQUIRY.
    FETCH FIRST PART USING PART_ID
       ON ERROR
       DISPLAY "PART " PART_ID " NOT IN DATABASE"
       GO TO 100-GET-PART.
200-GET-SUPPLY-INFO.
    FETCH NEXT SUPPLY WITHIN PART_INFO
       AT END
```

```
      FETCH OWNER WITHIN PART_INFO
      GO TO 300-GET-QUOTE-INFO.
 ****************************************************
 * The FETCH OWNER statement resets currency to      *
 * point to the owner. This allows the search for    *
 * PR_QUOTE records to begin with the first member  *
 * record occurrence rather than after the           *
 * last SUPPLY record occurrence.                    *
 ****************************************************
    PERFORM 500-SUPPLY-ANALYSIS.
    GO TO 200-GET-SUPPLY-INFO.
300-GET-QUOTE-INFO.
    FETCH NEXT PR_QUOTE WITHIN PART_INFO
       AT END
       GO TO 100-GET-PART.
    PERFORM 600-QUOTE-ANALYSIS.
    GO TO 300-GET-QUOTE-INFO.
```

Figure 5.13 is an occurrence diagram of a forked set. The figure shows a part record owning five PART_INFO member records.

Figure 5.13. Forked Set Occurrence



5.9.2. Multiset Relationships

A set cannot contain an owner record and a member record of the same type. Nor can a simple set represent a many-to-many relationship. To simulate such relationships, Oracle CODASYL DBMS uses the concept of multiset relationships. Multiset relationships occur when two set types share a common record type called a junction record. The junction record can contain information specific to the relationship. An empty junction record contains only pointer information used by the DBCS to establish the multiset relationship. This section discusses three kinds of multiset relationships:

- Many-to-many relationships between two types of records
- Many-to-many relationships between records of the same type
- One-to-many relationships between records of the same type

5.9.2.1. Many-to-Many Relationships Between Two Types of Records

To build a many-to-many relationship between two types of records, the DBA uses a junction record. For example, a part can be supplied by many vendors, and one vendor can supply many parts. The SUPPLY record type in Figure 5.14 links or joins PART records with VENDOR records.

Figure 5.14. Bachman Diagram of a Many-to-Many Relationship Between Two Types of Records

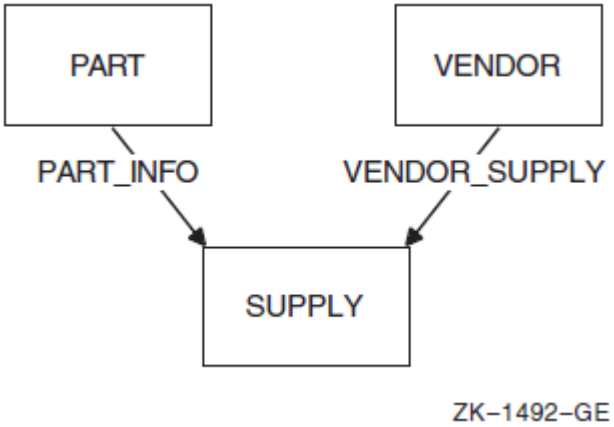
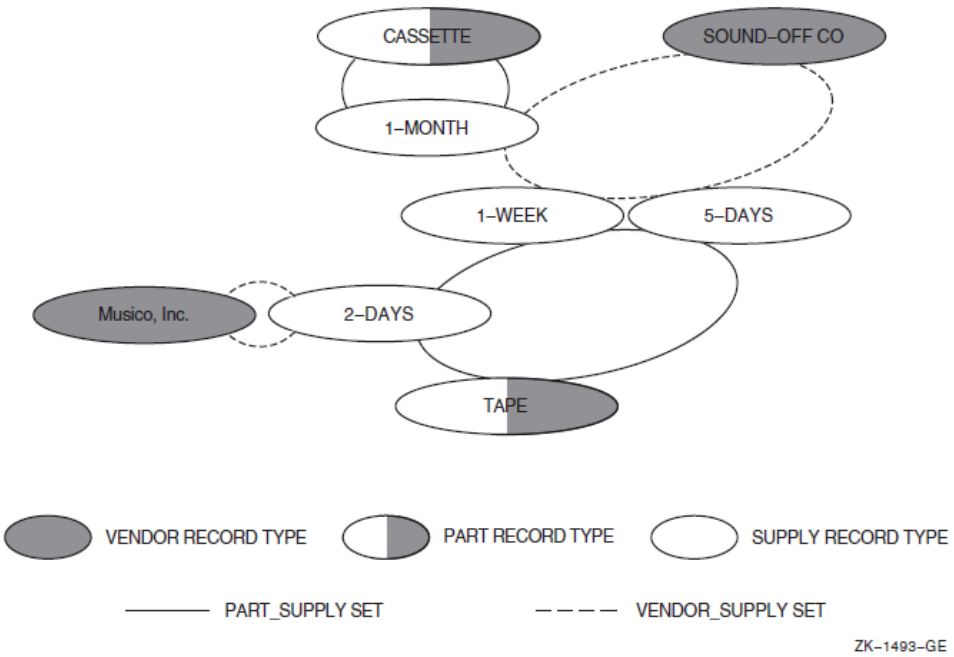


Figure 5.15 is an occurrence diagram of a many-to-many relationship between two types of records. This diagram typifies a many-to-many relationship because it shows a part (TAPE) being supplied by more than one vendor and a vendor (SOUND-OFF CO.) supplying more than one part. You could add additional vendors for a part by joining new supply records to a part and its new vendors. You could also add additional parts supplied by one vendor by joining supply records to the vendor and the new parts.

Figure 5.15. Many-to-Many Relationship Between Two Types of Records



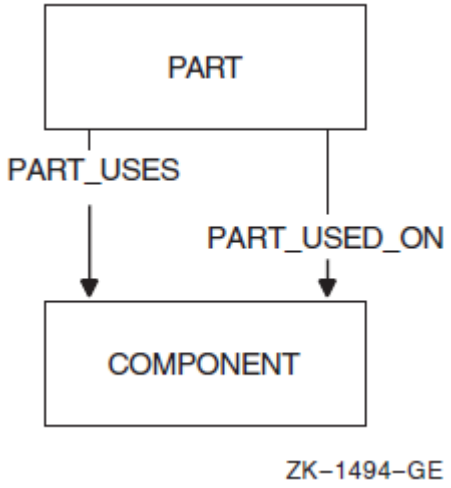
5.9.2.2. Many-to-Many Relationships Between Records of the Same Type

To represent a relationship between record occurrences of the same type, the DBA builds a many-to-many relationship using member records to create the necessary links. Figure 5.16 shows a many-to-

many relationship between records of the same type, where PART is the owner of both PART_USES and PART_USED_ON set types and COMPONENT is the junction record.

PART_USES is a bill of materials set type that links a PART owner record through its COMPONENT member records to the part's subassemblies. The link to the subassemblies is from COMPONENT member records up to the PART_USED_ON set type and back to PART owner records.

Figure 5.16. Bachman Diagram of a Many-to-Many Relationship Between Records of the Same Type



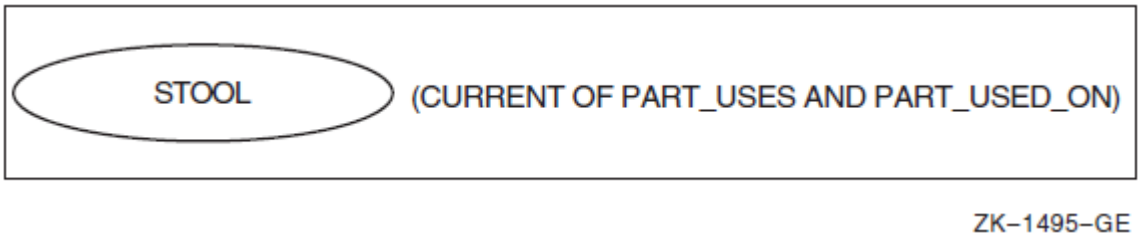
For example, assume you are creating a bill of materials and you have a finished part, a stool, made from one stool seat and four stool legs. Figure 5.17, Figure 5.18, Figure 5.19, and Figure 5.20 show occurrence diagrams of the bill of materials you would need to build a stool.

To complete the bill of materials you have to link the stool seat and stool legs to the finished part, the stool. You would:

1. Use the FIND statement to locate the stool.

```
PROCEDURE DIVISION.
100-FIND-STOOL.
    MOVE "STOOL" TO PART_DESC.
    FIND FIRST PART USING PART_DESC.
```

Figure 5.17. Current of PART_USES and PART_USED_ON

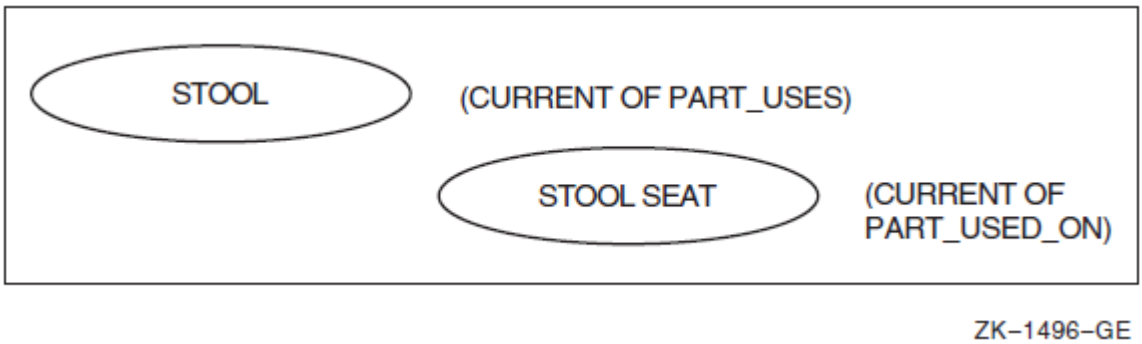


2. Use the FIND statement to locate the stool seat retaining PART_USES currency. Because PART usually owns both sets, using a FIND or FETCH statement to locate PART changes both set currency indicators. Retaining PART_USES currency keeps a pointer at STOOL; otherwise, STOOL SEAT would be current for both sets. Section 5.13 discusses currency indicators in more detail.

```
200-FIND-STOOL-SEAT.
```

```
    MOVE "STOOL SEAT" TO PART_DESC.
    FIND FIRST PART USING PART_DESC
               RETAINING PART_USES.
```

Figure 5.18. Retain PART_USES Currency

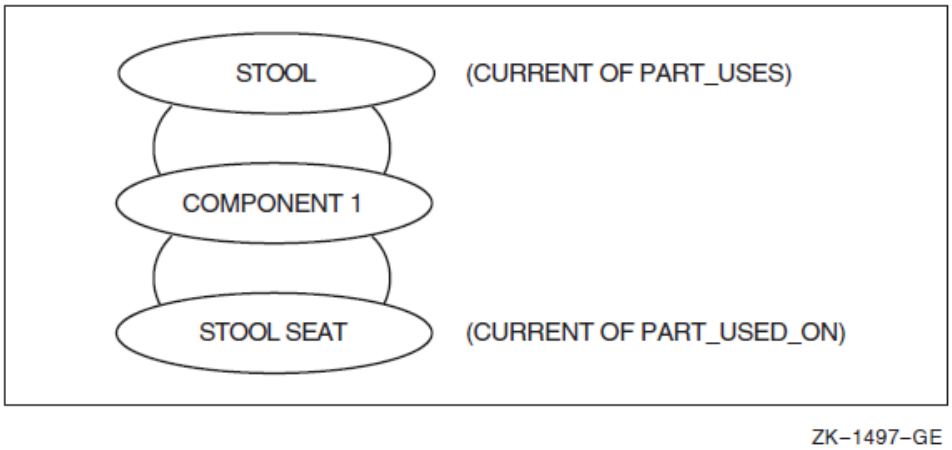


3. Build a COMPONENT record (component 1), and store it retaining PART_USES currency. Because COMPONENT participates in the PART_USES set, storing it normally changes the set's currency. Therefore, executing a STORE statement with the retaining clause keeps STOOL as current of PART_USES. At this point, STOOL is the PART_USES owner of component 1, and STOOL SEAT is the PART_USED_ON owner of component 1.

   Since the insertion mode for COMPONENT is automatic in both set types, a STORE COMPONENT automatically connects COMPONENT to both set types.

```
300-CONNECT-COMPONENT-1.     MOVE 1 TO COMP_QUANTITY.     STORE
COMPONENT RETAINING PART_USES.
```

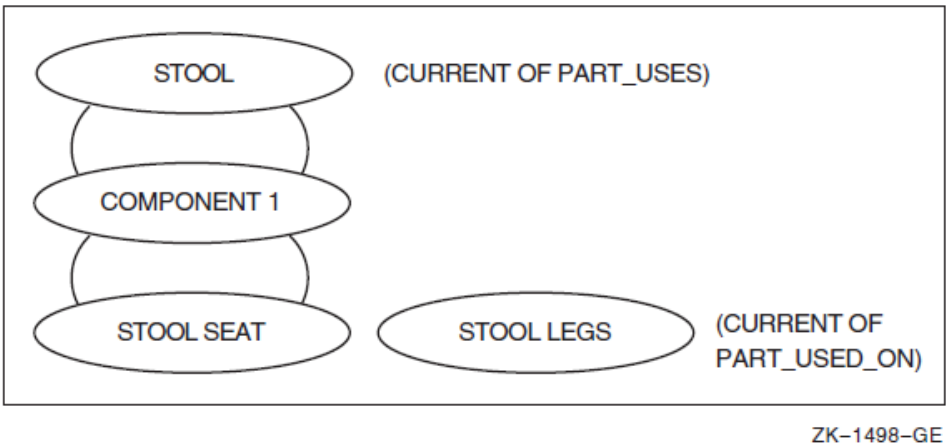
Figure 5.19. COMPONENT Is Connected to Both Set Types



4. Use the FIND statement to locate the stool legs, again retaining PART_USES currency, thus keeping STOOL current of PART_USES.

```
400-FIND-STOOL-LEGS.
    MOVE "STOOL LEGS" TO PART_DESC.
    FIND FIRST PART USING PART_DESC
               RETAINING PART_USES.
```

Figure 5.20. Finding the Stool Legs While Keeping STOOL Current of PART_USES



5. Build a second COMPONENT record (component 4) and store it. This links both PART_USES owner STOOL and PART_USED_ON owner STOOL LEGS to component 4. This completes all the necessary relationships you need to create the bill of materials shown in Figure 5.21.

```
500-CONNECT-COMPONENT-4.
    MOVE 4 TO COMP_QUANTITY.
    STORE COMPONENT.
```

Figure 5.21. Completed Bill of Materials

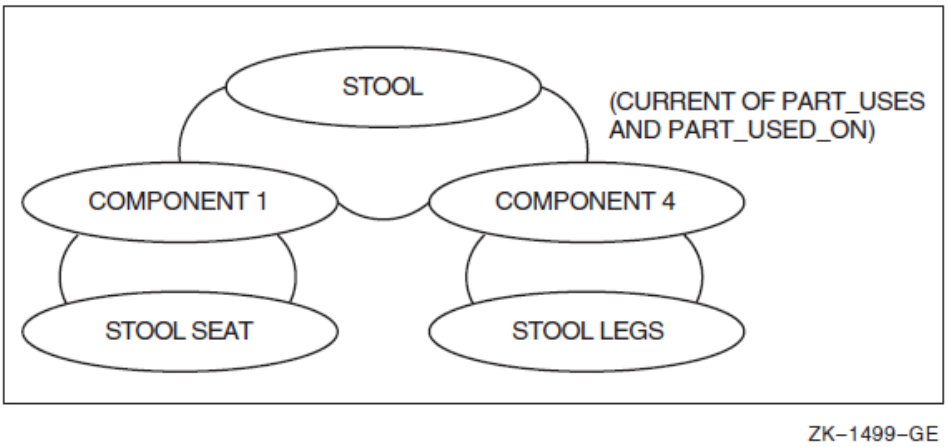
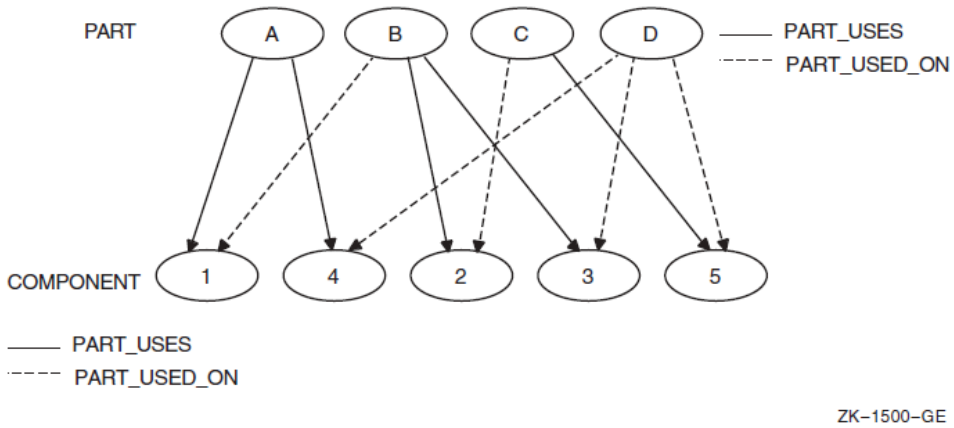


Figure 5.22 shows the relationship between PART records and COMPONENT records. The solid lines connect PART_USES owners to their members and the dotted lines connect PART_USED_ON owners to their members.

Figure 5.22. Occurrence Diagram of a Many-to-Many Relationship Between Records of the Same Type



The STOOL program in Example 8.5 loads and connects the parts for the STOOL bill of materials presented earlier in this section. It uses the relationship represented in Figure 5.16 to print its parts breakdown report in Section 8.6. Example 5.3 explains how to read the parts breakdown report.

Example 5.3. Sample Parts Breakdown Report

```
          PARTS BREAKDOWN REPORT
PART A (Part A information)
     PART B (Part B information)
          PART C (Part C information)
               PART D (Part D information)
          PART D (Part D information)
     PART D (Part D information)
```

The sample parts breakdown report shows that:

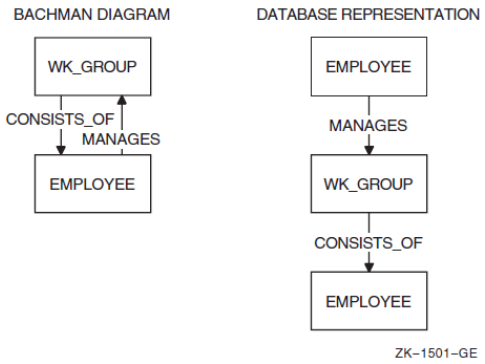
- PART A is built using two subassemblies: PART B and PART D.
- PART B is built using PART C and PART D.
- PART C is built using PART D.

5.9.2.3. One-to-Many Relationships Between Records of the Same Type

To build a one-to-many relationship between records of the same type, the DBA uses junction records. In a one-to-many relationship between records of the same type, either record type can be the junction record. However, in Figure 5.23 the WK_GROUP record type serves as the junction record because the EMPLOYEE record type has most of the relationship's data.

The record type EMPLOYEE includes all employees – supervisors, managers, and so forth. A manager can have many supervisors and a supervisor can have many employees. Conversely, an employee can have only one supervisor, and a supervisor can have only one manager.

Figure 5.23. One-to-Many Relationship Between Records of the Same Type



To show a relationship between employees (that is, who works for whom), Figure 5.23 uses the record type WK_GROUP as a link to establish an owner-to-member relationship. For example, a manager or supervisor would own a WK_GROUP record occurrence in the MANAGES set, and the same WK_GROUP occurrence owns any number of EMPLOYEE records in the CONSISTS_OF set. The relationship would be as follows: one occurrence of EMPLOYEE owns a WK_GROUP record occurrence, which in turn owns zero or more occurrences of the EMPLOYEE record type.

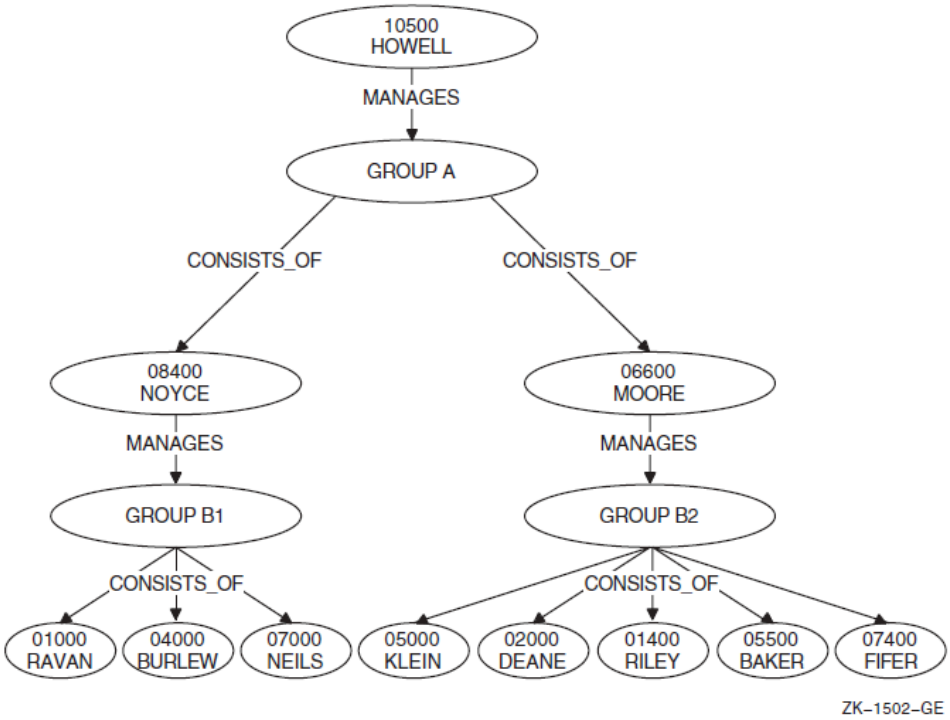
A one-to-many relationship between records of the same type is different from a many-to-many relationship between records of the same type because:

- An employee can have only one manager, while a part can be used on many subassemblies.
- The EMPLOYEE record type can participate both as an owner and a member in its relationship with WK_GROUP.
- The PART record type can participate only as an owner in its relationship with COMPONENT.

Example 8.6 shows how to use DML for hierarchical relationships. The example uses the diagram in Figure 5.23.

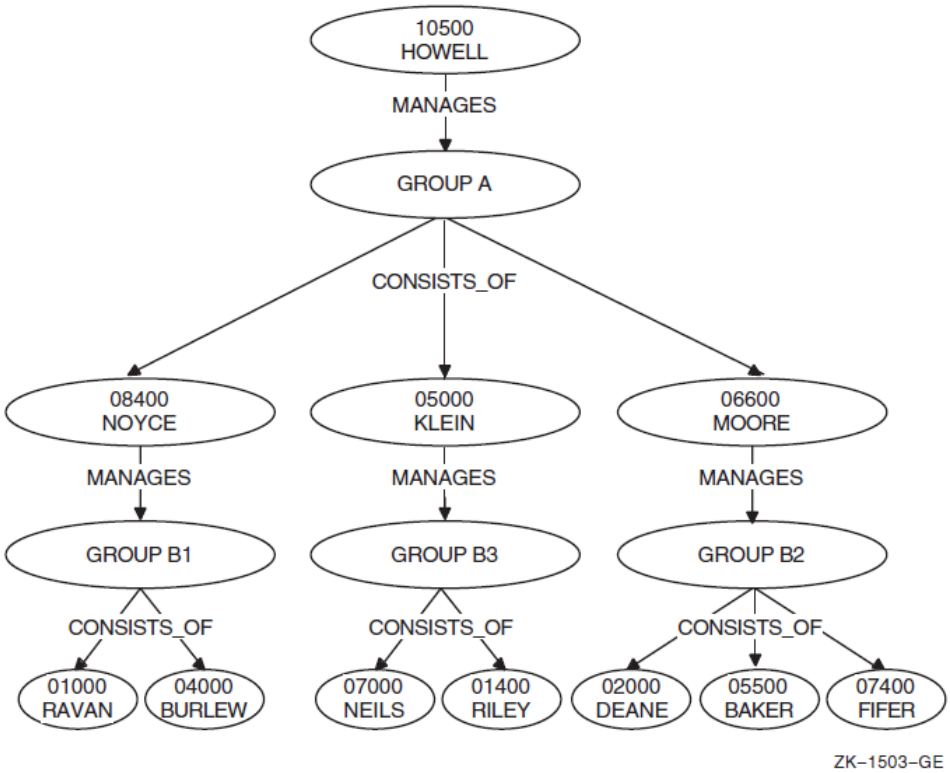
The data in Figure 5.24 shows sample EMPLOYEE records and the connecting WK_GROUP links (Groups A, B1, and B2). For example, employee Howell manages a group that consists of employees Noyce and Moore.

Figure 5.24. Sample Data Prior to Update



Assume that employee Klein is promoted to supervisor with Neils and Riley reassigned to work for him. Figure 5.24 shows the relationship between EMPLOYEE and WK_GROUP record types prior to the update, and Figure 5.25 shows the relationship after the update.

Figure 5.25. Sample Data After Update



Example 8.6 (PERSONNEL-UPDATE program) uses the data in Figure 5.24 and shows you how to:

1. Load the database (PERSONNEL-UPDATE).
2. Display the contents of the database on your terminal using the Report Writer before changing relationships (PERSONNEL-REPORT) (see Figure 5.24 and Example 8.7).
3. Create new relationships (PROMOTION-UPDATE).
4. Display the contents of the database on your terminal using the Report Writer after changing relationships (PERSONNEL-REPORT) (see Figure 5.25 and Example 8.8).

5.10. Areas

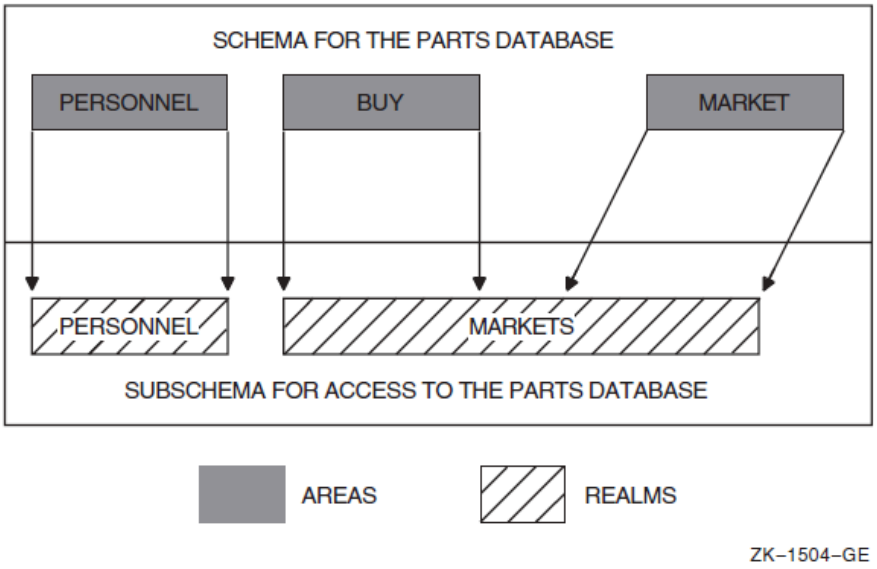
The DBA divides the database into areas so you can reference the database in sections instead of an entire unit. Areas are physical divisions of the database that are defined in the schema and are used to dump selectively, verify, or recover sections of the database; improve I/O; group logically related record types; and provide protection restrictions. Areas are stored as separate files and can be on separate volumes.

5.11. Realms

A realm is a group of one or more areas. Realms are logical divisions of the database. A realm is the object of the DML READY statement. Figure 5.26 shows the relationship between the schema, areas, subschema, and realms. Even though realms can contain data from more than one area, the type of data they contain is dependent on the subschema. It acts as a filter, allowing access to only specific data items.

Entire realms, as well as individual database records, are locked by the DBCS as they are retrieved by the run unit, and the degree of locking depends on the specific DML command used. For more information, see Section 6.1.

Figure 5.26. Database Relationships



5.12. Run Unit

The term run unit and program are not the same. A run unit is an executable image that may access a database, while a program can be used in two or more run units. For example, program SHOW-

EMPLOYEE can be run simultaneously by a payroll department employee to obtain employee data, and by an accountant to obtain job cost data. Each person controls his or her own run unit.

5.13. Currency Indicators

When you access database records, the database control system (DBCS) uses pointers called currency indicators to keep track of record storage and retrieval. VSI COBOL for OpenVMS uses currency indicators to remember records and their positions in the database. Currency indicators can be changed by DML statement execution. Thus, they assist in defining the environment of a DML statement and are updated as a result of executing DML statements.

One currency indicator exists for each realm, set type, and record type defined in your subschema. Another currency indicator, called the run-unit currency indicator, also exists for the run unit.

All the currency indicators in a run unit are null prior to execution of the first DML statement. The null value indicates there is neither a current record nor a current position. Execution of certain DML statements can change the value of currency indicators. However, currency indicators do not change if statement execution fails.

The DBCS also uses currency indicators as place markers to control its sequence of access to the database. For example, if VENDOR is the name of the vendor records in Figure 5.2, then the current of VENDOR is normally the vendor record most recently accessed. Likewise, in the set VENDOR_SUPPLY, the current of VENDOR_SUPPLY is normally the most recently accessed record of that set. Note that current of set could be either a member or owner record because both record types are part of the VENDOR_SUPPLY set.

Failure to establish correct currency can produce incorrect or unpredictable results. For example, you might unknowingly modify or delete the wrong record. The following sections describe how the DBCS sets currency indicators and how to use currency status in a DML program.

5.13.1. Current of Realm

Each realm currency indicator can be null or it can identify:

- A record and its position in the realm
- A position in the realm but not a specific record

A record identified by the realm currency is called current of realm. The DBCS updates current of realm only when you reference a different record within the realm. For example:

```
000100 PROCEDURE DIVISION.
000110 .
000120 .
000130 .
000500 FIND FIRST PART WITHIN BUY.
000510 FIND FIRST PART WITHIN MAKE.
000520 FIND NEXT PART WITHIN BUY.
000600 FIND NEXT SUPPLY WITHIN PART_INFO.
000610 .
000620 .
000630 .
```

For example, if LABEL and CASSETTE are in the BUY realm, while TAPE is in the MAKE realm, statement 000500 sets the first occurrence of PART record in realm BUY (LABEL) as current of realm

BUY. Statement 000510 sets the first occurrence of PART record in realm MAKE (TAPE) as current of realm MAKE. Notice that current of realm BUY is still the record occurrence accessed in statement 000500. Statement 000520 changes the current of realm BUY to the next occurrence PART record in realm BUY (CASSETTE). Current of realm MAKE remains the record accessed in statement 000510. Because the SUPPLY record type is located in the MARKET realm, statement 000600 sets the current of MARKET realm to the first SUPPLY record in the current PART_INFO set.

5.13.2. Current of Set Type

Each set type currency indicator can be null or it can identify:

- A record and its position in the set type
- A position in the set type but not a record

A record identified by a set type currency indicator is the current record for the set type, or current of set type.

If the ordering criterion for a set type is NEXT or PRIOR, the set type's currency indicator specifies the insertion point for member records. Therefore, if the currency indicator points to an empty position, a member record can be inserted in the specified position. If the currency indicator points to a record and NEXT is specified, a member record can be inserted after the current record for the set type. If the currency indicator points to a record and PRIOR is specified, a member record can be inserted before the current record for the set type.

The DBCS updates current of set type only when you reference a record that participates either as an owner or member in a set type occurrence. For example:

```
000100 PROCEDURE DIVISION.
.
.
.
000500 FIND FIRST PART.
000510 FIND FIRST SUPPLY WITHIN PART_INFO.
000520 FIND OWNER WITHIN VENDOR_SUPPLY.
000600
.
.
.
```

Statement 000500 sets the first occurrence of PART (LABEL) as current of set types PART_USES, PART_USED_ON, and PART_INFO. This is because PART records participate in three sets (see Figure 5.2). Because LABEL is current of PART_INFO, statement 000510 sets the first occurrence of SUPPLY (4-DAYS) owned by LABEL as current of set type PART_INFO. Because SUPPLY also participates in the VENDOR_SUPPLY set, this statement also sets the current occurrence of SUPPLY as current of set type VENDOR_SUPPLY. Statement 000520 sets the VENDOR owner record occurrence (SOUND-OFF CO.), which owns the current SUPPLY record, as current of set type VENDOR_SUPPLY.

5.13.3. Current of Record Type

Each record type currency indicator can be null or it can identify:

- A record and its position among other records of the same type

- A position among records of the same type, but not identify a record

Record type currency indicators do not identify a record type's relationship with other record types.

A record identified by a record type currency indicator is called current of record type. The DBCS updates the current of record type only when you reference a different record occurrence of the record type. References to other record types do not affect this currency. For example:

```
000100 PROCEDURE DIVISION
.
.
.
000500 FIND LAST PART.
000510 FIND FIRST SUPPLY WITHIN PART_INFO.
000520 FIND NEXT WITHIN PART_INFO.
000530 FIND FIRST VENDOR.
.
.
.
```

Statement 000500 sets the last occurrence of PART (TAPE) as current of record type PART. Statement 000510 sets the SUPPLY record occurrence (2-DAYS) as current of record type SUPPLY. Statement 000520 updates current of record type for SUPPLY to record occurrence (1-WEEK). Statement 000530 sets VENDOR record occurrence (MUSICO INC.) as current of record type VENDOR.

5.13.4. Current of Run Unit

The Database Control System (DBCS) updates the currency indicator for current of run unit each time a run unit refers to a different record occurrence, regardless of realm, set, or record type. For example:

```
000100 PROCEDURE DIVISION.
.
.
.
000500 FIND FIRST PART.
000510 FIND FIRST SUPPLY.
000520 FIND FIRST VENDOR.
000600 .
000610 .
000620 .
```

Statement 000500 sets the current of run unit to the first PART record occurrence (LABEL). Statement 000510 then sets the first SUPPLY record occurrence (4-DAYS) as current of run unit. Finally, statement 000520 sets the first VENDOR record (MUSICO INC.) as current of run unit. The first VENDOR record occurrence remains current of run unit until the run unit refers to another record occurrence.

5.14. Currency Indicators in a VSI COBOL for OpenVMS DML Program

Currency indicators are the tools you use to navigate through a database. Because of the many set relationships a database can contain, touching a record with a DML statement often changes more than one currency indicator. For example, a FETCH to a set type record can change currency for the set type, the record type, the realm, and the run unit. Knowing currency indicator status, how currency indicators change, and what statements control them, will help you locate the correct data.

Example 5.4 searches for TAPE vendors with a supply rating equal to A. Assume that record TAPE resides in BUY realm and that the SUPPLY record occurrences 2-DAYS and 5-DAYS have a SUP_RATING equal to A. Figure 5.27 shows how DML statements affect currency status.

Example 5.4. Currency Indicators

```
000100 PROCEDURE DIVISION
.
.
.
000490 100-FETCH-THE-PART.
000500 MOVE "TAPE" TO PART_DESC.
000510 FETCH FIRST PART USING PART_DESC.
000520 MOVE "A" TO SUP_RATING.
000550 200-FIND-SUPPLY.
000560 FIND NEXT SUPPLY WITHIN PART_INFO
000570 USING SUP_RATING.
000580 AT END
000590 GO TO 500-NO-MORE-SUPPLY.
000600 FETCH OWNER WITHIN VENDOR_SUPPLY.
000610 ************************
000620 * VENDOR PRINT ROUTINE *
000630 ************************
000640 GO TO 200-FIND-SUPPLY.
```

Statement 000500 provides the search argument used by statement 000510. Statement 000510 fetches the first occurrence of PART with a PART_DESC equal to TAPE. Statement 000520 provides the search argument used by statement 000560. Statement 000560 finds each member record occurrence of SUPPLY with a SUP_RATING equal to A owned by the PART with a PART_DESC equal to TAPE.

If, instead of its present structure, statement 000560 read “FIND NEXT SUPPLY USING SUP_RATING,” the search for supply records would not be restricted to supply member records in the PART_INFO set owned by TAPE. Instead, the search would extend to all supply records, finding all vendors with a supply rating equal to A, who may or may not be suppliers of TAPE.

Figure 5.27. Currency Status by Executable DML Statement

STATEMENT	RUN UNIT	REALM			SET TYPE		RECORD		
		MARKET	MAKE	BUY	PART_INFO	VENDOR_SUPPLY	PART	VENDOR	SUPPLY
510	CASSETTE	NULL	NULL	CASSETTE	CASSETTE	NULL	CASSETTE	NULL	NULL
*560	2–DAYS	2–DAYS	NULL	CASSETTE	2–DAYS	2–DAYS	CASSETTE	NULL	2–DAYS
*600	MUSICO INC.	MUSICO INC.	NULL	CASSETTE	2–DAYS	MUSICO INC.	CASSETTE	MUSICO INC.	2–DAYS
**560	5–DAYS	5–DAYS	NULL	CASSETTE	5–DAYS	5–DAYS	CASSETTE	MUSICO INC.	5–DAYS
**600	SOUND–OFF	SOUND–OFF	NULL	CASSETTE	5–DAYS	SOUND–OFF	CASSETTE	SOUND–OFF	5–DAYS

* First execution
** Second execution

ZK–1505–GE

5.14.1. Using the RETAINING Clause

You use the RETAINING clause to save a currency indicator you want to refer to. You use the RETAINING clause to: (1) navigate through the database and return to your original starting point, or

(2) walk through a set type. (The expression “walk through a set type” implies a procedure where you access all owner records and their respective members.) Refer to the Section 4.10, Section 5.14.1, and Section 6.11 for further information.

After finding all members for an owner, the current of run unit is the last accessed member record occurrence in the set. If the next statement is a FIND NEXT for an owner, you may not retrieve the next owner. This is because:

- Current of set type (in this case, the last member record occurrence) is also current of run unit.
- Without a WITHIN clause, the FIND (or FETCH) is based on current of run unit.

Because DBCS uses currency status as pointers, a FIND NEXT VENDOR WITHIN MARKET uses current of MARKET realm to find the next owner record occurrence. To make sure a FIND (or FETCH) next owner statement finds the next logical owner record, use the RETAINING clause, as shown in Example 5.5.

Example 5.5. Using the RETAINING Clause

```
000100 PROCEDURE DIVISION.
.
.
.
000400 100-VENDOR-SUPPLY-WALKTHRU.
000410 FETCH NEXT VENDOR WITHIN MARKET
000420 AT END GO TO 900-ALL-DONE.
.
.
.
************************
* VENDOR PRINT ROUTINE *
************************
.
.
.
000500 300-GET-VENDORS-SUPPLY.
000510 FETCH NEXT SUPPLY WITHIN VENDOR_SUPPLY
000520 RETAINING REALM
000530 AT END
000540 GO TO 100-VENDOR-SUPPLY-WALKTHRU.
.
.
.
************************
* SUPPLY PRINT ROUTINE *
************************
.
.
.
000550 GO TO 300-GET-VENDORS-SUPPLY.
```

Statement 000410 fetches the vendors. Statement 000510 fetches the supply records owned by their respective vendors. Statement 000510 also uses the RETAINING clause to save the realm currency.

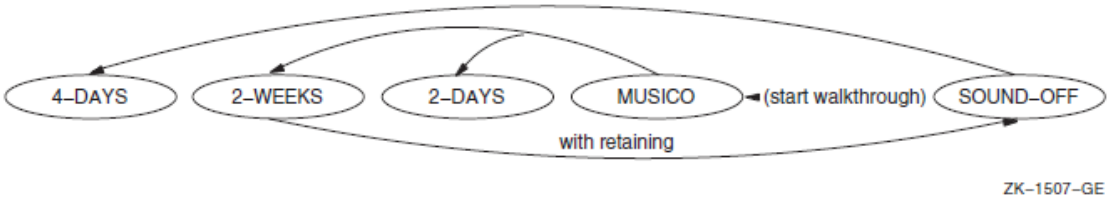
A FETCH NEXT SUPPLY (statement 000510) without the RETAINING clause makes SUPPLY current for the run unit, its record type, all sets in which it participates, and its realm. When SUPPLY record 2-WEEKS in Figure 5.28 is current of run unit, a FETCH NEXT VENDOR statement fetches the

vendor whose physical location in the database follows the 2-WEEKS record. As shown in Figure 5.28, MUSICO would be the next vendor and the program would be in an infinite loop.

Figure 5.28. Physical Representation of a Realm Without a RETAINING Clause

A FETCH NEXT SUPPLY with the RETAINING clause makes SUPPLY current for the run unit and the set types but keeps the vendor record current for the realm shown in Figure 5.29. By retaining the realm currency when you fetch supply records, the last accessed vendor record remains current of realm. A FETCH NEXT VENDOR WITHIN MARKET statement uses the realm currency pointer, which points to MUSICO to fetch the next vendor, SOUND-OFF. Therefore, retaining the realm currency allows you to fetch the next logical vendor record.

Figure 5.29. Physical Representation of a Realm with a RETAINING Clause



5.14.2. Using Keeplists

A keeplist is a stack of database key values (see the description of KEEPLIST in LD (Keeplist Description)). The KEEP and FIND ALL statements build a stack of keys that lets you retrieve Oracle CODASYL DBMS records using the ordinal position of the stack entries. Oracle CODASYL DBMS calls the table of entries a keeplist. Each execution of the KEEP or FIND ALL statement adds a record's database key (dbkey) value to the end of a keeplist and places a retrieval lock on the record. Therefore, other users cannot change a record while its database key is in your keeplist.

You can use a keeplist to retain the database key of a record after that record is no longer current. That is, by inserting a database key into a keeplist, you can continue to reference that record by specifying the keeplist name and database key value in your DML statement. This is especially useful when you want to remember a record during a long sequence of DML commands that affect currency, or when you want to remember a list of records.

A keeplist can contain zero, one, or several database key values. To activate a keeplist, use the KEEP statement. To empty a keeplist, use the FREE statement. All keeplists are deallocated when you execute a COMMIT or ROLLBACK unless COMMIT RETAINING is used.

The following example adds database keys to a keeplist.

```
000100 PROCEDURE DIVISION.
.
.
.
000140 100-KEEPLIST-EXAMPLE.
```

```
000150 FETCH FIRST VENDOR.
000160 KEEP CURRENT USING KEEPLIST-1.
000170 FETCH FIRST SUPPLY WITHIN VENDOR_SUPPLY.
000180 FETCH OWNER WITHIN PART_INFO.
000190 IF PART_STATUS = "M"
000200 KEEP CURRENT WITHIN VENDOR_SUPPLY USING KEEPLIST-1.
```

Statement 000160 adds the vendor record's dbkey value (the current of run unit) to KEEPLIST-1. Figure 5.30 shows the contents of KEEPLIST-1 after execution of statement 000160. Adding a record's database key to a keeplist also prevents record updating by other concurrent users. Statements 000190 and 000200 add a supply record's database key to KEEPLIST-1 whenever its PART_INFO owner has a status of M. Figure 5.31 shows the contents of KEEPLIST-1 after the execution of statements 000190 and 000200.

Figure 5.30. State of KEEPLIST-1 After Executing Line 000160

KEEPLIST–1	
Database Key (DBKEY)	ORDINAL POSITION
vendor dbkey	1

ZK–6063–GE

Figure 5.31. State of KEEPLIST-1 After Executing Lines 000190 and 000200

KEEPLIST–1	
Database Key (DBKEY)	ORDINAL POSITION
vendor dbkey	1
supply dbkey	2

ZK–6064–GE

You can use database key values as search arguments to locate database records. For example:

```
FIND 2 WITHIN KEEPLIST-1
```

This statement:

- Uses the value of the number 2 to locate the ordinal position of a database key value
- Uses the database key value to find a record

The KEEP statement can also transfer database key values from one keeplist to another. For example:

```
KEEP OFFSET 2 WITHIN KEEPLIST-1 USING KEEPLIST-2
```

This statement copies the second-positioned database key value in KEEPLIST-1 to the end of KEEPLIST-2.

The FREE statement removes database key value entries from a keeplist. For example:

```
FREE ALL FROM KEEPLIST-1
```

This statement removes all the entries from KEEPLIST-1.

You can remove keeplist entries by identifying their ordinal position within the keeplist. For example:

```
FREE 5 FROM KEEPLIST-2
```

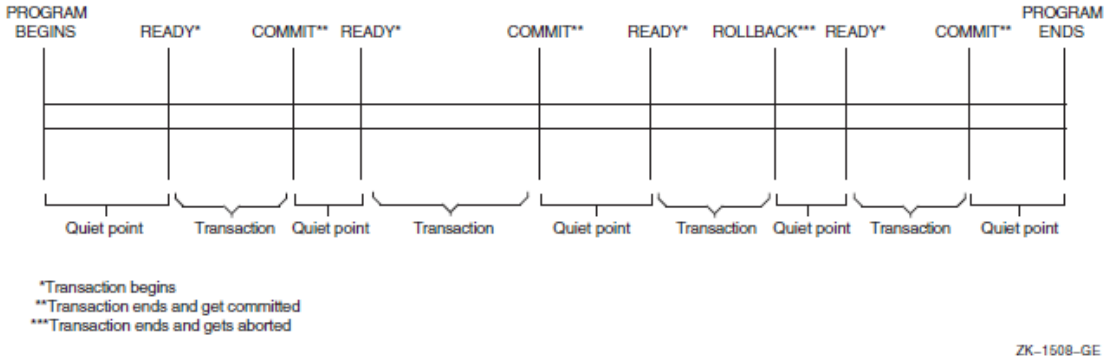
This statement removes the fifth-positioned database key value from KEEPLIST-2. Removing a keeplist entry changes the position of all the following entries. For example, after freeing entry 5, entry 6 becomes the fifth-positioned entry, entry 7 becomes the sixth-positioned entry, and so forth. The FREE statement changes the ordinal position of a database key value in the keeplist, not its contents.

5.14.3. Transactions and Quiet Points

You generally segment your run unit into transactions, bounded instances of run-unit activity. A transaction begins with the first DML statement in the run unit or with a READY statement that follows a COMMIT or ROLLBACK statement; continues through a series of DML data access statements; and ends with either a COMMIT statement, a ROLLBACK statement, or the termination of the run unit. Before the initial READY statement is issued, after the COMMIT or ROLLBACK, and before the next READY, the run unit is at a quiet point. A quiet point is the time that exists between the last executed COMMIT or ROLLBACK statement and the next READY statement, or the time prior to the first executed READY statement.

The Quiet Point – Transaction – Quiet Point continuum provides the DBCS with a structure that allows it to control access to and ensure the integrity of your data. To implement this control, the DBCS uses currency indicators and locking. Figure 5.32 shows the segmentation of a run unit into transactions and quiet points.

Figure 5.32. Transactions and Quiet Points



Chapter 6. DML Programming — Tips and Techniques

We've gathered some tips and techniques you can use to improve program performance and reduce development and debugging time. These include special use of modes, indicators, conditions and statements, as well as debugging techniques.

6.1. The Ready Modes

Proper use of the READY usage modes can improve system performance.

You inform the DBCS of your record-locking requirements when you issue the READY command. The command takes the form:

```
READY <allow-mode> <access-mode>
```

or

```
READY <access-mode> <allow-mode>
```

The allow- and the access-mode arguments pass your requirements to the DBCS.

The allow-mode object of the READY command indicates what you will allow other run units to do while your run unit works with storage areas within the realm you readied. There are four different allow modes as follows:

CONCURRENT	Permits other run units to ready the same realm or realms that contain the same storage areas as the realms your run unit readied. CONCURRENT also allows other run units to perform any DML function on those storage areas, including updates.
PROTECTED	Permits other run units to use the same storage areas as your run unit, but does not allow those run units to update records in the storage areas.
EXCLUSIVE	Prohibits other run units from even reading records from the restricted storage areas.
BATCH	Allows concurrent run units to update the realm. BATCH also allows you to access or update any data in the realm while preventing concurrent run units from accessing or updating the realm.

While the allow mode says what your run unit will allow other run units to do, the access mode says that your run unit will either read or write records (RETRIEVAL or UPDATE).

Because the UPDATE access mode can lock out other users, use it only for applications that perform database updates. If an application accesses the database for inquiries only, use the RETRIEVAL access mode. The RETRIEVAL mode also prevents a run unit from accidentally updating the database.

The combination of the allow mode and the access mode is called the usage mode. There are eight READY usage modes as follows:

- CONCURRENT RETRIEVAL

- CONCURRENT UPDATE
- PROTECTED RETRIEVAL (the system default)
- PROTECTED UPDATE
- EXCLUSIVE RETRIEVAL
- EXCLUSIVE UPDATE
- BATCH RETRIEVAL
- BATCH UPDATE

Use the CONCURRENT usage modes for applications requiring separate run units to simultaneously access the database. They allow other run units to perform a READY statement on your realm, and possibly change or delete the database records in that realm.

Use the PROTECTED usage modes only when unrestricted access might produce incorrect or incomplete results. Protected access prevents other run units from making changes to the data in your realm. However, run units in RETRIEVAL mode can still access (read-only) your realm.

Use the EXCLUSIVE usage modes only when you want to lock out all other users. The EXCLUSIVE mode speeds processing for your run unit and prevents other run units from executing a READY statement on your realm. When you specify EXCLUSIVE access, use only the realms you need. Eliminating the use of unnecessary realms minimizes lockout. Use the EXCLUSIVE allow mode to get the best performance from a single run-unit application. Care must be taken, however, because other run units are locked out and must wait for the exclusive run unit to finish before it can begin operations.

Use the BATCH RETRIEVAL usage mode for concurrent run units to update the realm. Use the BATCH UPDATE usage mode to access or update any data in the realm while preventing concurrent run units from accessing or updating the realm.

For more information on READY usage mode conflicts, see READY. It summarizes the effects of usage mode options on run units readying the same realms.

6.1.1. Record Locking

Concurrent run units can reference realms that map to the same storage area; the same records can be requested by more than one transaction at the same time. If two different transactions were allowed to modify the same data, that data would be rendered invalid. Each modification to the original data would be made in ignorance of other modifications, and with unpredictable results. Oracle CODASYL DBMS preserves the integrity of data shared by multiple transactions. It also provides levels and degrees of record locking. You can control access to, or lock:

- All records in a realm you intend to access
- Individual records as they are retrieved by DML statements

You can also lock records totally or allow some retrieval functions.

Record locking begins with the execution of the first READY statement in the run unit. At that time the DBCS is told of your storage area locking requirements. If you specify EXCLUSIVE allow mode, no

other run unit is allowed to access records in the specified realms. This is all the locking that the DBCS need do. If you specify CONCURRENT or PROTECTED modes, the DBCS initiates locking at the record level.

Individual records are locked as they are retrieved by the run unit. The degree of locking depends on the specific DML command used. For example, if your run unit executes a FETCH or FIND statement, the DBCS sets a read-only record lock, allowing other run units to read, but not update, the records. This lock is also set if your run unit assigns the database key associated with the record to a keeplist with the KEEP verb. (Note if you use FETCH or FIND FOR UPDATE, a no-read lock is placed on the specified record).

As a record is retrieved, the lock is held at this level until there are no more currency indicators pointing to the record. If the program assigns a record to a keeplist, the lock is held by your run unit until it frees the record from the keeplist with a FREE statement. However, if a currency indicator points to a record whose database key is also in a keeplist, then a FREE statement to that keeplist entry still leaves the read-only lock active for that record. Similarly, if the same database key is in several keeplists, then freeing it from one keeplist does not release the other read-only locks.

However, the DBCS grants a no-read access lock if your run unit specifies a DML update verb, such as STORE, CONNECT, or MODIFY. Your run unit retains the lock on this record until the change is committed to the database by the DML COMMIT verb or the change is terminated or canceled by ROLLBACK.

The Run-Time System notifies the DBCS each time a run unit requests a locked record, thus keeping track of which records are locked and who is waiting for which records. This logging helps the DBCS determine whether a conflict exists, such as multiple run units requesting, but not being allowed, to access or change the same record. For more information on record locking, refer to the Oracle CODASYL DBMS documentation on database design and programming.

6.2. COMMIT and ROLLBACK

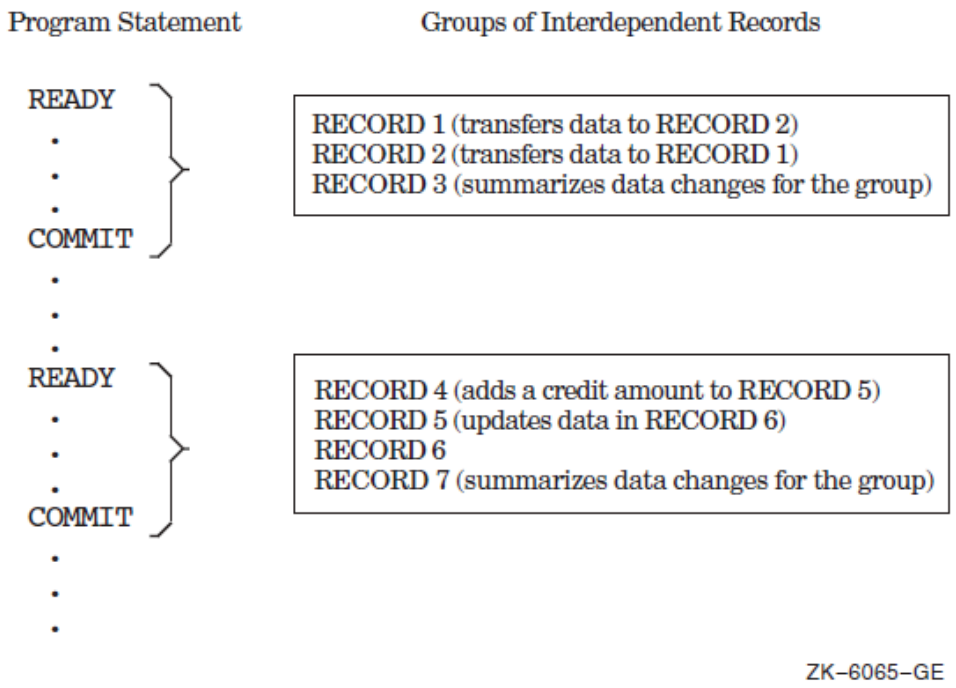
When you are in CONCURRENT UPDATE mode, any changes made to a record lock the record and prevent its access by other run units. For example, if a program updates 200 customer records in one transaction, the 200 customer records are unavailable to other run units. To minimize lockout, use the COMMIT statement as often as possible.

The COMMIT statement makes permanent all changes made to the database, frees all locks, and nulls all currencies. It also establishes a quiet point for your run unit.

The RETAINING clause can be used with the COMMIT statement. COMMIT RETAINING does not empty keeplists; retains all currency indicators; does not release realm locks; demotes no-read locks to read-only locks; then releases locks for all records except those in currency indicators or keeplists and makes visible any changes made to the database.

To use COMMIT properly, you need to know about application systems. For example, you might want to execute a COMMIT each time you accomplish a logical unit of work. Or, if you were updating groups of interdependent records like those in Figure 6.1, you would execute a COMMIT only after updating a record group.

Figure 6.1. Using the COMMIT Statement



The ROLLBACK statement cancels all changes made to the database since the last executed READY statement and returns the database to its condition at the last quiet point. The DBCS performs an automatic ROLLBACK if your run unit ends without executing a COMMIT or if it ends abnormally.

In Example 6.1 an order-processing application totals all items ordered by a customer. If the order amount exceeds the credit limit, the program executes a ROLLBACK and cancels the transaction updates. Notice that the credit limit is tested for each ordered item, thus avoiding printing of an entire invoice prior to cancelling the order.

Example 6.1. ROLLBACK Statement

```
        .
        .
        .
READY-UPDATE.
   READY TEST_REALM CONCURRENT UPDATE.
 **************************
 * FETCH CUSTOMER ROUTINE *
 **************************
   .
   .
   .
 ******************************
 * FETCH ORDERED ITEMS ROUTINE *
 ******************************
   .
   .
   .
CREDIT-LIMIT-CHECK.
    MULTIPLY ORDERED-QUANTITY BY UNIT-PRICE
             GIVING ORDER-AMOUNT.
    ADD ORDER-AMOUNT TO TOTAL-AMT.
    IF TOTAL-AMT IS GREATER THAN CUST-CREDIT-LIMIT
```

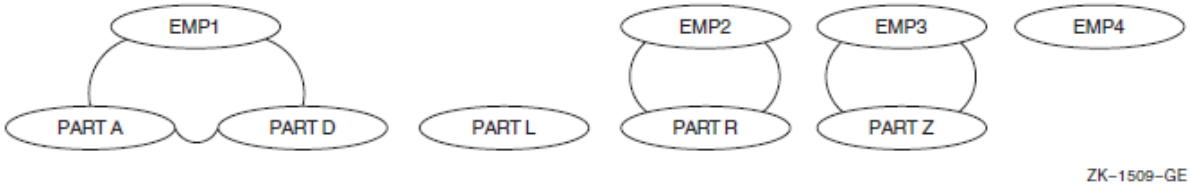
```
            ROLLBACK
            PERFORM CREDIT-LIMIT-EXCEEDED
        ELSE PERFORM PRINT-INVOICE-LINE.
```

6.3. The Owner and Member Test Condition

The FIND OWNER statement finds the owner of the current of set type, which may not be the same as the current of run unit. Thus, executing a FIND OWNER WITHIN set-name when the current of run unit record is not connected to the specified set returns the owner of the member that is current of set type.

Figure 6.2 shows occurrences of the RESPONSIBLE_FOR set type where employees are responsible for the design of certain parts.

Figure 6.2. Occurrences of the RESPONSIBLE_FOR Set Type



Example 6.2 uses the data in Figure 6.2 to perform an analysis of PART D, PART L, and the work of the engineer responsible for each part. The set retention class is optional.

Example 6.2. Owner and Member Test Condition

```
            .
            .
            .
000130 MAIL-LINE ROUTINE.
000140     MOVE "PART D" TO PART_DESC.
000150     PERFORM FIND-PARTS.
000160     MOVE "PART L" TO PART_DESC.
000170     PERFORM FIND-PARTS.
000180     GO TO ALL-FINISHED.
000190 FIND-PARTS.
000200     FIND FIRST PART USING PART_DESC.
000210     IF PART-IS-MISSING
000220             PERFORM PART-MISSING.
000230     PERFORM PARTS-ANALYSIS.
000240     FIND OWNER WITHIN RESPONSIBLE_FOR.
000250     PERFORM WORKLOAD-ANALYSIS.
000250 DONE-ANALYSIS.
000260     EXIT.
            .
            .
            .
```

When PART L becomes current of run unit, a FIND OWNER (statement 000240) finds PART D's owner, thus producing incorrect results. This is because a FIND OWNER WITHIN set-name uses the current of set type and PART L is not a member of any RESPONSIBLE_FOR set type occurrence. To prevent this error, statement 000240 should read:

```
IF RESPONSIBLE_FOR MEMBER
```

```
     FIND OWNER WITHIN RESPONSIBLE_FOR
ELSE
     PERFORM PART-HAS-NO-OWNER.
```

6.4. Using IF EMPTY Instead of IF OWNER

The OWNER test condition does not test whether the current record owns any member records. Rather, this condition tests if the current record participates as an owner record. If a record type is declared as the owner of a set type, an OWNER test for that record type will always be true. Therefore, referring to Figure 6.2, if EMP4 is the object of an IF RESPONSIBLE_FOR OWNER test, the result is true because EMP4 is an owner record, even though the set occurrence is empty.

To test if an owner record owns any members, use the EMPTY test condition. For example:

```
IF RESPONSIBLE_FOR IS EMPTY PERFORM EMPTY-ROUTINE
        ELSE ...
```

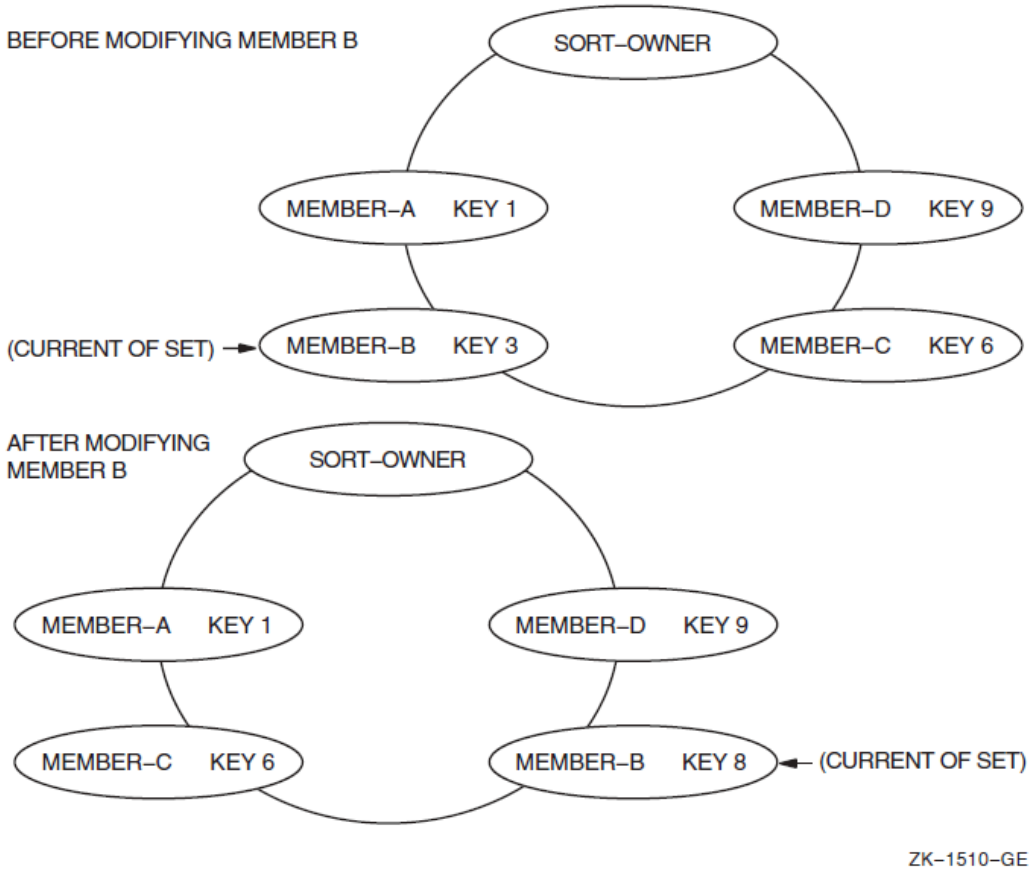
Thus, if EMP4 is the object of an IF RESPONSIBLE_FOR IS EMPTY test, the result is true because the set occurrence has no members.

6.5. Modifying Members of Sorted Sets

If the schema defines a set's order to be SORTED and you modify any data items specified in the ORDER IS clause of the schema, the record may change position within the set occurrence. If the record does change position, the set's currency changes to point to the member record's new position.

Figure 6.3 shows a set occurrence for SORT_SET where MEMBER-B's key (KEY 3) was changed to KEY 8. Before altering the record's key, the set currency pointed to MEMBER-B, and a FETCH NEXT MEMBER WITHIN SORT_SET fetched MEMBER-C. However, the modification to MEMBER-B's key repositions the record within the set occurrence. Now, a FETCH NEXT MEMBER WITHIN SORT_SET fetches the MEMBER-D record.

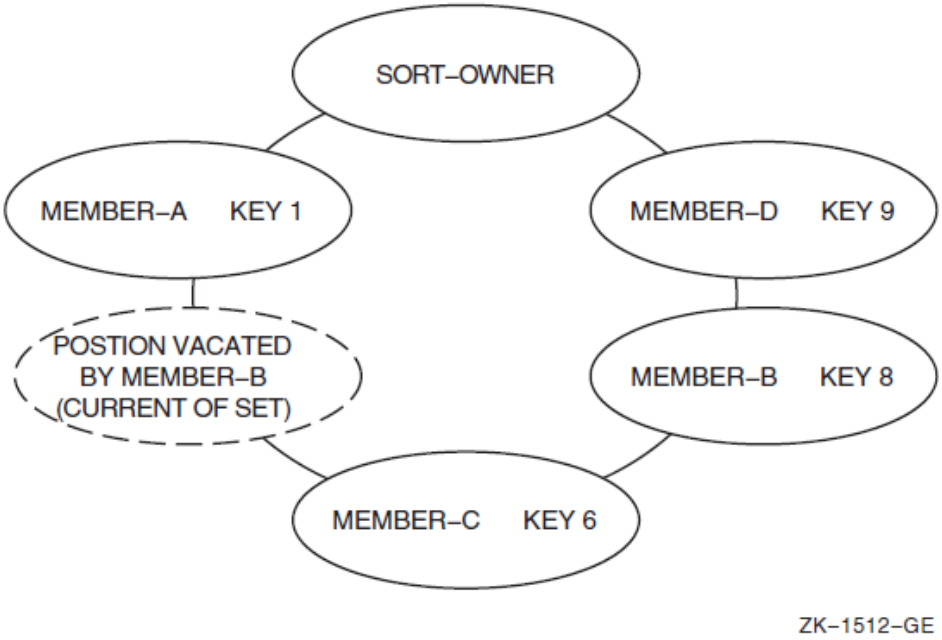
Figure 6.3. Modifying Members of Sorted Sets



When you change the contents of a data item specified in the ORDER IS SORTED clause and you do not want the set's currency to change, use the RETAINING clause with the MODIFY statement. Thus, MODIFY repositions the record and RETAINING keeps the currency indicator pointing at the position vacated by the record. Figure 6.4 shows how the following example retains currency for SORT_SET.

```
FETCH NEXT WITHIN SORT_SET.
IF MEMBER_KEY = "KEY 3"
   MOVE "KEY 8" TO MEMBER_KEY
   MODIFY MEMBER_KEY RETAINING SORT_SET.
```

Figure 6.4. After Modifying MEMBER_B and Using RETAINING



If MEMBER_B's key was changed to KEY 4, the record's position in the set occurrence would not change, and a FETCH NEXT WITHIN SORT_SET would fetch MEMBER_C.

6.6. CONNECT and DISCONNECT

When the set membership class is MANUAL, use the CONNECT statement to link a member record to its set occurrence. You can also use CONNECT for AUTOMATIC sets, provided that the retention class is OPTIONAL and you have disconnected the record.

When you use the CONNECT statement, specify the set or sets where the record is to be connected. Executing a CONNECT statement without the set list clause connects the record to all sets in which it can be, but is not yet, a member.

Before you execute a CONNECT statement, be sure that currency for the specified set type points to the correct set occurrence. If not, the member record will participate in the wrong set occurrence. (For more information on currency, see Section 5.13 and Section 5.14.) You cannot execute a CONNECT for a record that participates as an owner of the specified set.

If the set retention class is OPTIONAL, use the DISCONNECT statement to remove a member record from a specified set. The DISCONNECT statement does not delete a record from the database.

When you use the DISCONNECT statement, specify the sets from which the record will be disconnected. Executing a DISCONNECT without the set list clause disconnects the record from all the sets in which it participates as an optional member. You cannot execute a DISCONNECT for a record that participates as an owner of the specified set or that has a set retention class of FIXED or MANDATORY. Refer to the Section 4.7 for an explanation of how set membership class affects certain DML verbs.

6.7. RECONNECT

Use the RECONNECT statement to remove a member record from one set occurrence and connect it to another occurrence of the same set type, or to a different position within the same set. To transfer a member record:

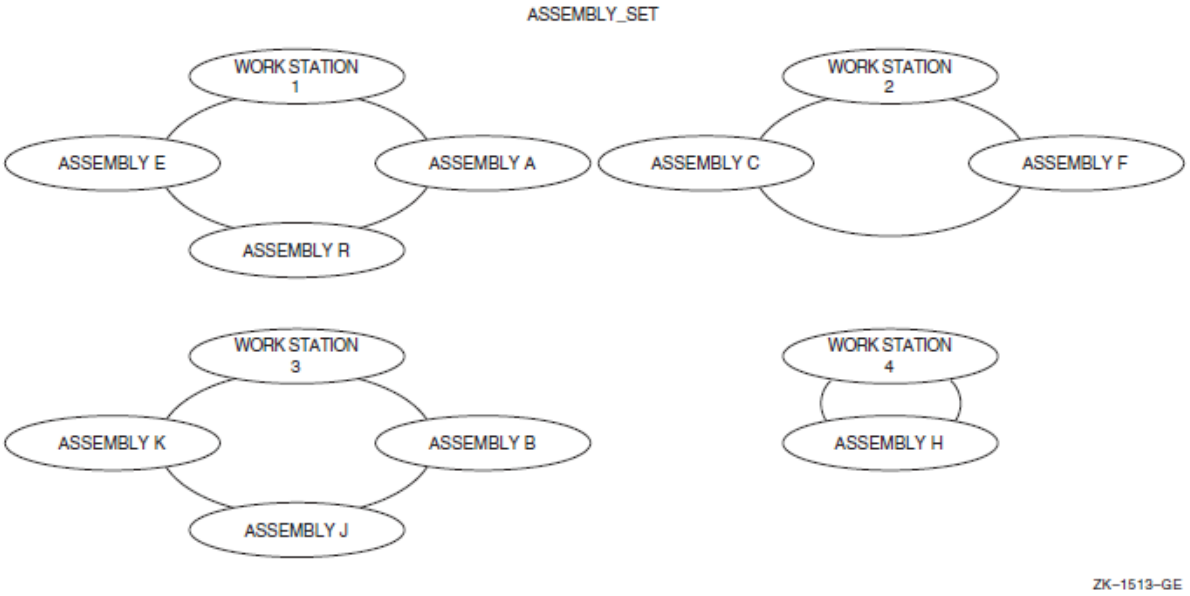
1. Use the FETCH (or FIND) statement to select a record in the set occurrence. This can be either a member or an owner of the set occurrence you want to connect to.

2. Use the FETCH (or FIND) statement with the RETAINING clause to transfer the member record you want. This keeps the currency for the targeted record.

3. Execute a RECONNECT statement using the WITHIN clause.

The RECONNECT statement is useful in applications such as production control where manufactured items move down an assembly line from one work station to another. In Figure 6.5, work stations are the owner records and assemblies are the member records.

Figure 6.5. Occurrence Diagram Prior to RECONNECT



Example 6.3 transfers ASSEMBLY R, a machine base, to WORK STATION 2 for electrical assembly. The order of insertion is LAST.

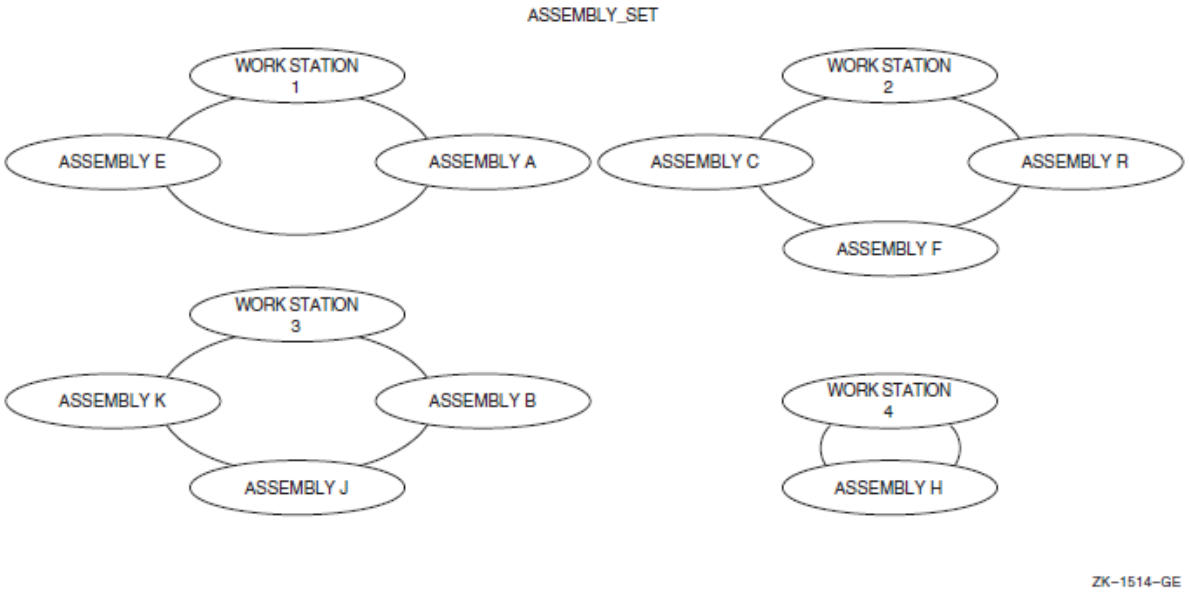
Example 6.3. RECONNECT Statement

```
     .
     .
     .
GET-WORK-STATION.
    MOVE 2 TO WORK_STATION_ID.
    FIND FIRST WORK_STATION USING WORK_STATION_ID.
    MOVE "R" TO ASSEMBLY_ID.
    FIND FIRST ASSEMBLY USING ASSEMBLY_ID
         RETAINING ASSEMBLY_SET.
 *******************************************************
 * The RETAINING clause retains work station 2 as       *
 * current of ASSEMBLY_SET. Otherwise, the found member  *
 * would be current of set and the RECONNECT would fail. *
```

```
*******************************************************
   RECONNECT ASSEMBLY WITHIN ASSEMBLY_SET.
   .
   .
   .
```

Figure 6.6 shows the ASSEMBLY_SET after execution of the RECONNECT statement. Notice the ASSEMBLY A record replaces the R record's position in the WORK STATION 1 set occurrence. Also, execution of the RECONNECT makes the ASSEMBLY R record current for the ASSEMBLY_SET.

Figure 6.6. Occurrence Diagram After RECONNECT



6.8. ERASE ALL

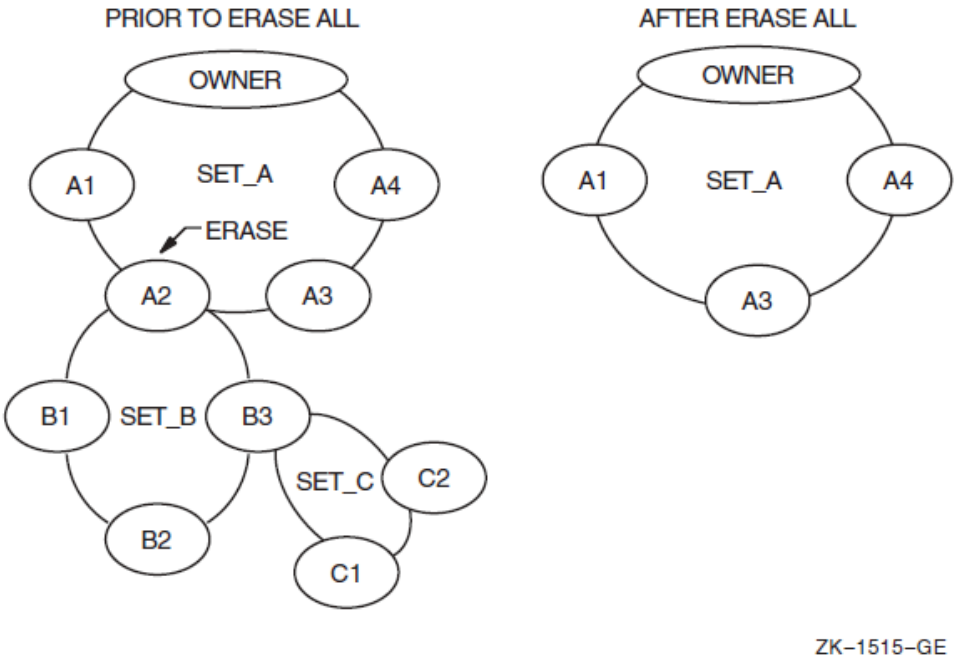
The ERASE statement deletes one or more records from the database. However, it can delete more than you intended. Accidental deletes can occur because of the ERASE statement's cascading effect. The cascading effect can happen whenever the erased record is the owner of a set. Thus, if the current record is an owner of a set type, an ERASE ALL deletes:

- The current record.
- All records in sets owned by the current record.
- Any records in sets owned by those members. Note that this is a repetitive process.

This is called a *cascading delete.*

The occurrence diagrams in Figure 6.7 show the results of using the ERASE ALL statement.

Figure 6.7. Results of an ERASE ALL



The ERASE ALL statement is the only way to erase an owner of sets with MANDATORY members.

6.9. ERASE Record-Name

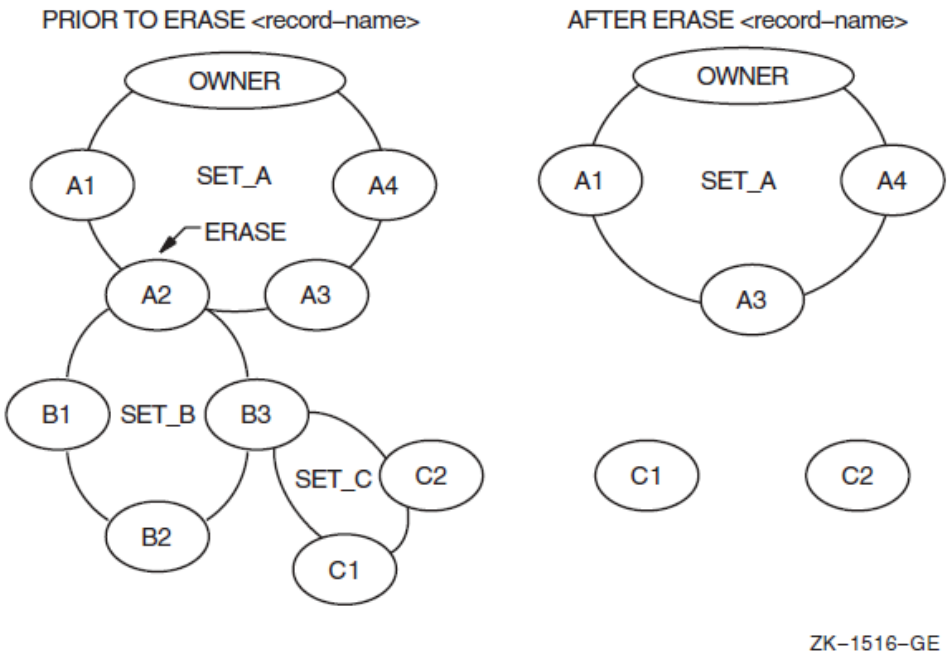
If you do not use the ERASE ALL statement but use the ERASE record-name, and the erased record is the owner of a set, the ERASE statement deletes:

- The current record.
- All FIXED members of sets owned by the current record.
- All FIXED members of sets owned by records in rule 2. Note that this is a repetitive process.

If the current record owns sets with OPTIONAL members, these records are disconnected from the set, but remain in the database.

The occurrence diagrams in Figure 6.8 show the results of using the ERASE record-name statement when affected members have an OPTIONAL set membership. In this figure, B records are FIXED members of the SET_B set and C records are OPTIONAL members of the SET_C set. Notice that records C1 and C2 are disconnected from the set, but remain in the database while B1 through B3 are erased.

Figure 6.8. Results of an ERASE Record-Name (with Both OPTIONAL and FIXED Retention Classes)



Remember, records removed from a set but not deleted from the database can still be accessed.

6.10. Freeing Currency Indicators

Use the FREE database-key-id statement to release the currency indicators for realms, records, sets, or the run unit. You use the FREE statement: (1) to establish a known currency condition before executing a program routine, and (2) to release record locks.

6.10.1. Establishing a Known Currency Condition

Establishing a known currency condition is helpful in many situations — for example, if you have a program that performs a customer analysis and prints three reports. The first report prints all customers with a credit rating greater than $1,000, the second report prints all customers with a credit rating greater than $5,000, and the third report prints all customers with a credit rating greater than $10,000. Because some customers will appear on more than one report, you want each report routine to start its customer analysis with the first customer in the database.

By using the FREE CURRENT statement at the end of a report routine, as shown in Example 6.4, you null the currency and allow the next print routine to start its analysis at the first customer.

Example 6.4. FREE CURRENT Statement

```
      .
      .
      .
MAIN-ROUTINE.
    READY TEST_REALM CONCURRENT RETRIEVAL.
    PERFORM FIRST-REPORT-HEADINGS.
    PERFORM PRINT-FIRST-REPORT THRU PFR-EXIT
            UNTIL AT-END = "Y".
    MOVE "N" TO AT-END.
```

```
    PERFORM SECOND-REPORT-HEADINGS.
    PERFORM PRINT-SECOND-REPORT THRU PSR-EXIT
            UNTIL AT-END = "Y".
    MOVE "N" TO AT-END.
    PERFORM THIRD-REPORT-HEADINGS.
    PERFORM PRINT-THIRD-REPORT THRU PTR-EXIT
            UNTIL AT-END = "Y".
    MOVE "N" TO AT-END.
      .
      .
      .
    STOP RUN.
PRINT-FIRST-REPORT.
    FETCH NEXT CUSTOMER_MASTER
          AT END FREE CURRENT
                 MOVE "Y" TO AT-END.
    IF AT-END = "N" AND
       CUSTOMER_CREDIT_RATING IS GREATER THAN 1000
           PERFORM PRINT-ROUTINE.
PFR-EXIT.
    EXIT.
PRINT-SECOND-REPORT.
    FETCH NEXT CUSTOMER_MASTER
          AT END FREE CURRENT
                 MOVE "Y" TO AT-END.
    IF AT-END = "N" AND
          CUSTOMER_CREDIT_RATING IS GREATER THAN 5000
          PERFORM PRINT-ROUTINE.
PSR-EXIT.
    EXIT.
PRINT-THIRD-REPORT.
    FETCH NEXT CUSTOMER_MASTER
          AT END MOVE "Y" TO AT-END.
    IF AT-END = "N" AND
       CUSTOMER_CREDIT_RATING IS GREATER THAN 10000
       PERFORM PRINT-ROUTINE.
PTR-EXIT.
    EXIT.
```

The FREE CURRENT statement in the PRINT-FIRST-REPORT paragraph nulls the default run-unit currency, thereby providing a starting point for the PRINT-SECOND-REPORT paragraph. The FREE CURRENT statement in the PRINT-SECOND-REPORT paragraph does the same for the PRINT-THIRD-REPORT paragraph. Thus, by nullifying the default run-unit currency, the FREE CURRENT statements allow the first execution of the FETCH NEXT CUSTOMER_MASTER statement to fetch the first customer master in TEST_REALM.

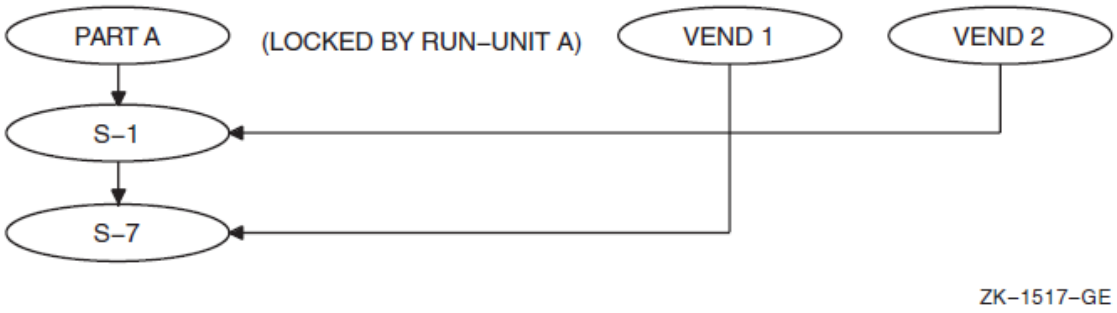
6.10.2. Releasing Record Locks

Regardless of the READY mode used, you always have a record lock on the current of run unit. Even the READY CONCURRENT RETRIEVAL mode locks the current record and puts it in a read-only condition. Furthermore, if you are traversing the database, the current record for each record type you touch with a DML statement is locked and placed in a read-only condition. Record locking prevents other users from updating any records locked by your run unit.

A locked record can prevent accessing of other records. Figure 6.9 shows PART A locked by run unit A. Assume PART A has been locked by a FETCH statement. If run unit B is in READY UPDATE

mode and tries to: (1) update PART A, and (2) find all of PART A's member records and their vendor owners, then run unit B is locked out and placed in a wait state. A wait state occurs when a run unit cannot continue processing until another run unit completes its database transaction. Because run unit B uses PART A as an entry point for an update, the lock on PART A also prevents access to PART A's member records and the vendor owners of these member records.

Figure 6.9. Record Locking



If a record is not locked by a STORE or a MODIFY statement, or the database key for the record is not in a keeplist, you can unlock it by using the FREE CURRENT statement. By using the FREE CURRENT statement, you reduce lockout and optimize processing for other run units.

6.11. FIND and FETCH Statements

The FIND and FETCH statements locate a record in the database and make that record the current record of the run unit. The FETCH statement also copies the record to the user work area (UWA), thus giving you access to the record's data. The FIND does not place a record in the UWA. However, if your only requirement is to make a record current of run unit, use the more efficient FIND statement. For example, use the FIND statement if you want to connect, disconnect, or reconnect without examining a record's contents.

6.12. FIND ALL Option

The FIND ALL statement puts the database key values of one or more records into a keeplist. (See the description of FIND ALL in Section 4.9 for syntax details).

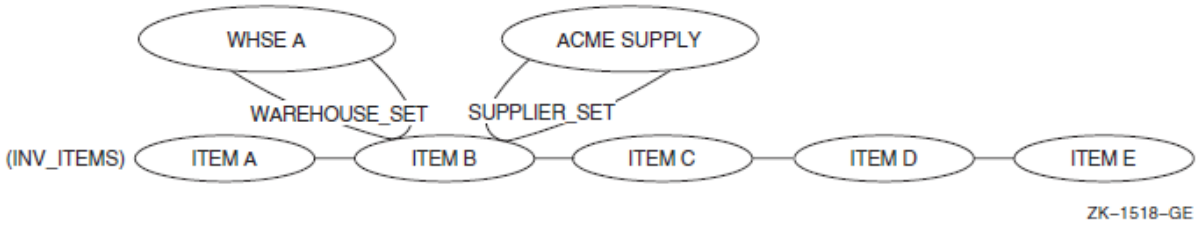
The following example locates all PART records with a PART_STATUS of J and puts their dbkey values in keeplist TWO.

```
FIND ALL TWO PART USING PART_STATUS
PART_STATUS X(1) = J
```

6.13. FIND NEXT and FETCH NEXT Loops

If you have a FIND NEXT or FETCH NEXT loop in your program, the first execution of the loop is the same as executing a FIND FIRST or FETCH FIRST. Unless you properly initialize them, currency indicators can affect selection of the specified record. For example, if ITEM B in Figure 6.10 is current for INV_ITEMS, a FIND NEXT INV_ITEMS makes ITEM C the current record for the run unit. You can null a currency by executing a FREE CURRENT statement.

Figure 6.10. Using FIND NEXT and FETCH NEXT Loops



Example 6.5 makes the INV_ITEMS currency null prior to executing a FETCH NEXT loop.

Example 6.5. FETCH NEXT Loop

```
     .
     .
     .
000100 GET-WAREHOUSE.
000110     MOVE "A" TO WHSE-ID.
000120     FIND FIRST WHSE_REC USING WHSE-ID.
000130 UPDATE-ITEM.
000140     MOVE "B" TO ITEM-ID.
000150     FETCH FIRST WITHIN WAREHOUSE_SET
000160          USING ITEM-ID.
     ****************************
     * INVENTORY UPDATE ROUTINE *
     ****************************
      .
      .
      .
     ****************************************************
     * The next statement nulls the run unit currency.   *
     * Therefore, the first execution of the FETCH NEXT  *
     * gets the first INV_ITEMS record.                  *
     ****************************************************
000170         FREE CURRENT.
000180 ANALYZE-INVENTORY.
000190     FETCH NEXT INV_ITEMS
000200           AT END GO TO END-OF-PROGRAM.
000210     GO TO ANALYZE-INVENTORY.
      .
      .
      .
```

You can also use FETCH NEXT and FIND NEXT loops to walk through a set type. Assume you have to walk through the WAREHOUSE_SET and reduce the reorder point quantity by 10 percent for all items with a cost greater than $500. Furthermore, you also want to check the supplier's credit terms for each of these items. You could perform the task as shown in Example 6.6

Example 6.6. Using a FETCH NEXT Loop to Walk Through a Set Type

```
     .
     .
     .
000100 FETCH-WAREHOUSE.
000110     FETCH NEXT WHSE_REC
000120           AT END PERFORM END-OF-WAREHOUSE
```

```
000130                  PERFORM WRAP-UP.
000140 ITEM-LOOP.
000150      FETCH NEXT INV_ITEM WITHIN WAREHOUSE_SET
000160            AT END
000170               FIND OWNER WITHIN WAREHOUSE_SET
000180               PERFORM FETCH-WAREHOUSE.
000190      IF INV_ITEM_COST IS GREATER THAN 500
000200         PERFORM SUPPLIER-ANALYSIS.
000210*     Reduce reorder point quantity by 10%.
000220      MODIFY INV_ITEM.
000230      GO TO ITEM-LOOP.
000240 SUPPLIER-ANALYSIS.
000250      IF NOT SUPPLIER_SET MEMBER
000260               DISPLAY "NO SUPPLIER FOR THIS ITEM"
000270               EXIT.
000280       FETCH OWNER WITHIN SUPPLIER_SET.
000290*      Check credit terms.
         .
         .
         .
```

Notice the FIND OWNER WITHIN WAREHOUSE_SET statement on line 000170. At the end of a WAREHOUSE_SET collection, statement 000170 sets the WAREHOUSE_SET type currency to the owner of the current occurrence. This allows the next execution of FETCH NEXT WHSE_REC to use current of record type WHSE_REC to find the next occurrence of WHSE_REC. Without statement 000170, a FETCH NEXT WHSE_REC would use the current of run unit, which is an INV_ITEM record type.

6.14. Qualifying FIND and FETCH

You can locate records by using the contents of data items as search arguments. You can use more than one qualifier as a search argument. For example, assume you want to print a report of all employees in department 5 with a pay rate of $7.50 per hour. You could use the department number as a search argument and use a conditional test to find all employees with a pay rate of $7.50. Or you could use both the department number and pay rate as search arguments, as follows:

```
    .
    .
    .
000500 SETUP-QUALIFIES.
000510     MOVE 5    TO DEPARTMENT-NUMBER.
000520     MOVE 7.50 TO EMPLOYEE-RATE.
000530     FREE CURRENT.
000540 FETCH-EMPLOYEES.
000550     FETCH NEXT EMPLOYEE
000560           USING DEPARTMENT-NUMBER EMPLOYEE-RATE
000570                 AT END GO TO EXIT-ROUTINE.
000580     PERFORM EMPLOYEE-PRINT.
000590     GO TO FETCH-EMPLOYEES.
    .
    .
    .
```

You can also locate records by using a WHERE clause to designate a conditional expression as a search argument. The following example fetches the first SUPPLY record whose SUP_LAG_TIME is 2 days or less.

```
000450 FETCH-SUPPLY.
000460       FETCH FIRST SUPPLY
000470             WITHIN PART_INFO
000480             WHERE SUP_LAG_TIME LESS THAN 2
000490             AT END GO TO EXIT-ROUTINE.
```

Chapter 7. Debugging and Testing VSI COBOL for OpenVMS DML Programs

The Database Query utility (DBQ) commands and generic DML statements are the tools you use to debug and test your VSI COBOL for OpenVMS program's DML statements. For example, you can use DBQ commands to display currency indicators, test program loops, or check your program's execution efficiency.

It is important to eliminate any logic errors prior to running a VSI COBOL for OpenVMS DML program against a live database, because poorly written or incorrect logic can corrupt a database. You can resolve some logic errors by desk-checking a program. Desk-checking involves reviewing the logical ordering and proper use of DML statements; for example, to check for executing a FIND when you intend to execute a FETCH, or executing a CONNECT instead of a RECONNECT. You can also use a debugger (refer to the debugging information in the *VSI COBOL User Manual* [https://docs.vmssoftware.com/vsi-cobol-for-openvms-user-guide/]). However, neither method gives you information on currency indicators and the effects DML statements have on them.

Another method of debugging VSI COBOL for OpenVMS DML programs is to test DML statements using the DBQ utility. DBQ is an online interactive utility that uses a split screen to show the results of each execution of a DML statement. It is also an effective database programming learning tool. For a complete description of the DBQ utility, refer to the Oracle CODASYL DBMS documentation on data manipulation and programming.

We recommend that you use all of these tools to design, test, and debug your VSI COBOL for OpenVMS DML programs.

7.1. DBQ Commands and DML Statements

The DBQ utility provides both generic DML statements and DBQ-specific commands. Generic DML statements are similar to the VSI COBOL for OpenVMS DML statements explained in Chapter 4. However, not all VSI COBOL for OpenVMS DML syntax is applicable to the DBQ utility. These statements and entries do not apply:

- SUB-SCHEMA SECTION
- LD statement
- AT END phrase
- ON ERROR phrase
- Scope terminators
- USE statement
- DB statement — Use the DBQ utility BIND command to identify the subschema you will use for testing and debugging. You cannot access a subschema until you bind it. If your program has this DB statement:

```
DB PARTSS3 WITHIN PARTS FOR NEW.
```

the comparable BIND statement is as follows:

```
dbq> BIND PARTSS3 FOR NEW
```

- ANY clause — The DBQ utility does not allow the ANY clause in a Record Selection Expression. Instead, use the FIRST clause.
- DUPLICATE clause — The DBQ utility does not allow the DUPLICATE clause in a Record Selection Expression. Instead, use the NEXT clause.
- WHERE clause — The operators of this clause are different.

For a complete discussion of generic DML, refer to the Oracle CODASYL DBMS documentation on data manipulation and programming.

7.2. Sample Debugging and Testing Session

This section shows how to use the DBQ utility for debugging and testing VSI COBOL for OpenVMS DML programs. Because the split screen limits the number of lines that can be displayed at one time, the split screen figures show the Bachman diagram only. Corresponding DBQ prompts, entries, and messages follow each Bachman diagram and are shown in their entirety.

The session tests and finds a logic error in the DML program statements in Example 7.1. The sample COBOL DML program is intended to:

1. Fetch the first PART in the database with a PART_ID equal to AZ177311
2. Fetch all SUPPLY records for the found PART
3. Check the PART's SUPPLY records for SUP_RATINGs equal to 0
4. Change all SUP_RATINGs equal to 0 to 5, and print SUPPLY records VENDOR_SUPPLY owners
5. Change PART's PART_STATUS to X if one or more of its SUPPLY records has a SUP_RATING equal to 5

Remember, the database key values displayed on your screen may be different from those in the examples.

Note

If you are currently accessing PARTSS3 with the DBQ utility and have made any changes to the database, use the ROLLBACK statement to cancel your changes. Otherwise, you might change the results of the debugging session.

Example 7.1. Sample VSI COBOL for OpenVMS DML Program Statements

The following DBQ session tests and debugs the sample DML program statements in Example 7.1:

```
$ DBQ
dbq> BIND PARTSS3 FOR NEW
dbq> READY PROTECTED UPDATE
```

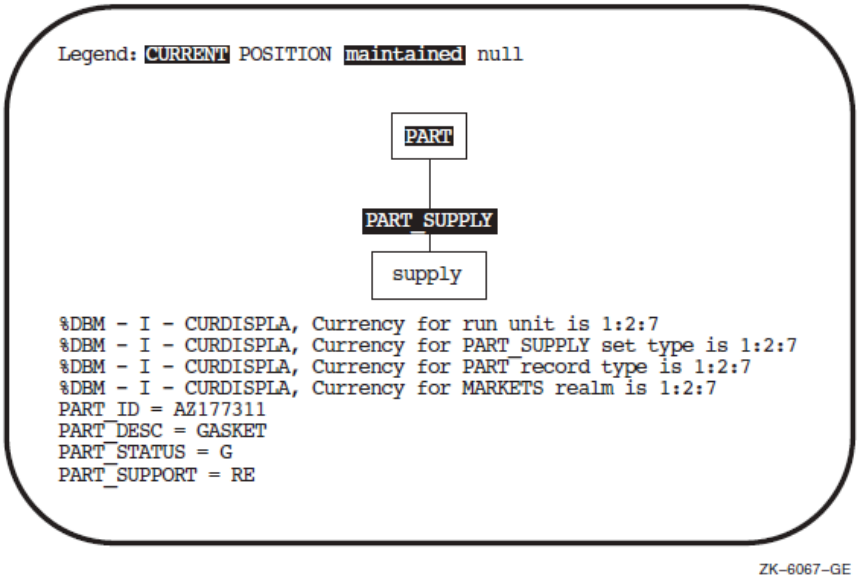
```
dbq> SET CURSIG
dbq> FETCH FIRST PART USING PART_ID
```

DBQ prompts you for a PART_ID value:

```
PART_ID [CHARACTER(8)] =AZ177311
```

Entering AZ177311 as the PART_ID value causes the Bachman diagram in Figure 7.1 to appear on your screen.

Figure 7.1. Split Screen After FETCH FIRST PART USING PART_ID

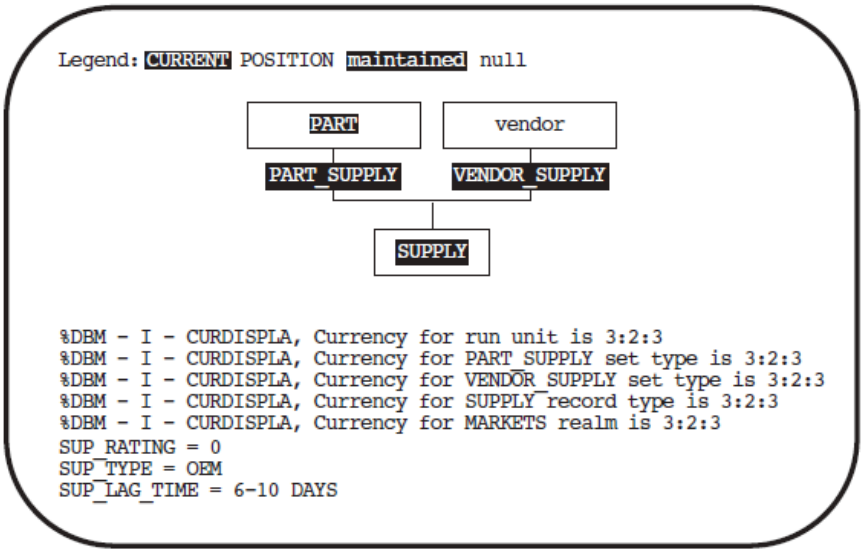


ZK-6067-GE

The next DML statement in Figure 7.2 is FETCH NEXT WITHIN PART_SUPPLY. Although this statement is in a performed loop, you can still test its logic by executing a series of FETCH NEXT WITHIN PART_SUPPLY until you find a SUP_RATING equal to 0.

```
dbq> FETCH NEXT WITHIN PART_SUPPLY
```

Figure 7.2. Split Screen After FETCH NEXT WITHIN PART_SUPPLY



ZK-6068-GE

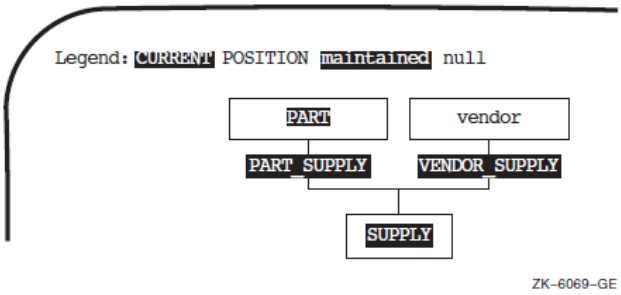
Because SUPPLY participates in two sets, the Bachman diagram in Figure 7.2 shows the set relationships for SUPPLY. Notice the SUPPLY record has a SUP_RATING equal to 0. Therefore, you can test the next DML statement.

```
dbq> MODIFY SUP_RATING
SUP_RATING [CHARACTER(1)]= 5
```

Notice how the MODIFY statement causes DBQ to issue a prompt, as shown in the preceding statement. When you MODIFY or STORE a record, DBQ prompts you for data entry by displaying the data name and its attributes. After entering the new SUP_RATING, use the RETURN key to execute the MODIFY statement.

Because this MODIFY statement does not change currency, the Bachman diagram in Figure 7.3 is the same as the one in Figure 7.2. Also, DBQ does not display currency update messages.

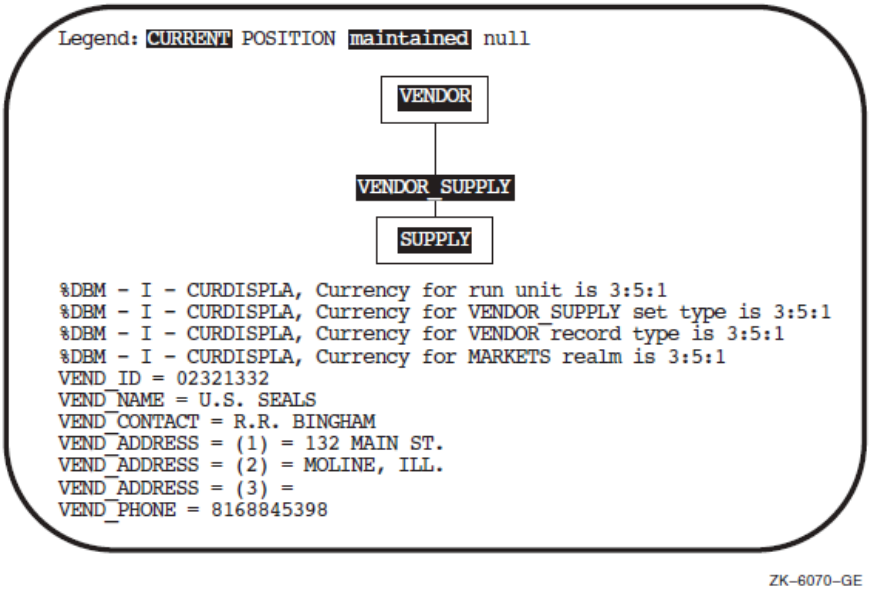
Figure 7.3. Split Screen After MODIFY SUP_RATING



The next statement to test is the FETCH for SUPPLY record's owner in the VENDOR_SUPPLY set.

```
dbq> FETCH OWNER WITHIN VENDOR_SUPPLY
```

Figure 7.4. Split Screen After FETCH OWNER WITHIN VENDOR_SUPPLY



Assuming the data item MODIFY-COUNT has a value 1, you can test the last (MODIFY PART) DML statement.

```
dbq> MODIFY PART_STATUS
PART_STATUS [CHARACTER(1)]= X
```

```
dbq>  DBM-F-WRONGRTYP, Specified record type not current record type
```

DBQ generates an error message indicating the MODIFY statement did not execute because the current of run unit is not a PART record. Comparing the shading and intensities of the Bachman diagram in Figure 7.4 with the legend shows the current record is a VENDOR record. Therefore, the diagram indicates that a MODIFY to the PART record will not work even before you attempt the MODIFY statement.

To correct the logic error, PART must be the current record type prior to execution of the MODIFY PART_STATUS statement. One way to correct the logic error is to execute a FETCH CURRENT PART statement before the MODIFY PART_STATUS statement. Example 7.2 shows a corrected version of the sample COBOL DML program statements in Example 7.1.

Example 7.2. Sample DML Program Statements

```
DATA DIVISION.
DB PARTSS3 WITHIN PARTS FOR NEW.
   .
   .
   .
PROCEDURE DIVISION.
000-BEGIN.
    READY PROTECTED UPDATE.
   .
   .
   .
    MOVE "AZ177311" TO PART_ID.
    FETCH FIRST PART USING PART_ID.
    MOVE "N" TO END-OF-COLLECTION.
    PERFORM A100-LOOP THROUGH A100-LOOP-EXIT
            UNTIL END-OF-COLLECTION = "Y".
   .
   .
   .
    STOP RUN.
A100-LOOP.
    FETCH NEXT WITHIN PART_SUPPLY
          AT END MOVE "Y" TO END-OF-COLLECTION
                 GO TO A100-LOOP-EXIT.
    IF SUP_RATING = "0"
          MOVE "5" TO SUP_RATING
          MODIFY SUP_RATING
          MOVE 1 TO MODIFY-COUNT
          FETCH OWNER WITHIN VENDOR_SUPPLY
          PERFORM PRINT-VENDOR.
    IF MODIFY-COUNT = 1
          MOVE "X" TO PART_STATUS
          MODIFY PART_STATUS.
A100-LOOP-EXIT.
    EXIT.
```

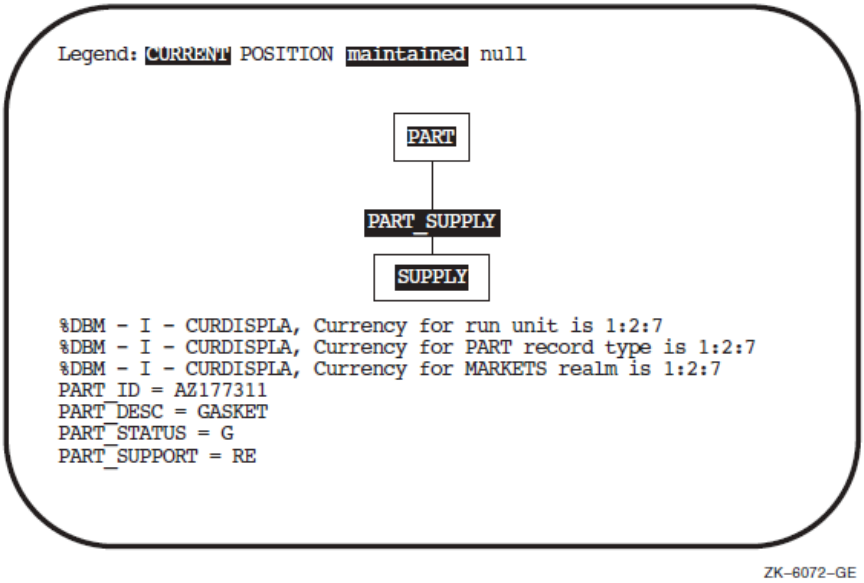
The FETCH CURRENT PART statement uses the RETAINING clause to keep the current SUPPLY record as current of PART_SUPPLY.

Continue testing, starting with the new FETCH statement.

```
dbq> FETCH CURRENT PART RETAINING PART_SUPPLY
```

Figure 7.5 shows that executing FETCH CURRENT PART RETAINING PART_SUPPLY makes PART the current record type, while the RETAINING clause keeps SUPPLY current of PART_SUPPLY set. Retaining the current supply record as current of PART_SUPPLY means the next execution of FETCH NEXT WITHIN PART_SUPPLY uses the current SUPPLY record's currency to locate the next SUPPLY record. If you executed a FETCH CURRENT PART without the RETAINING clause, a FETCH NEXT WITHIN PART_SUPPLY would use PART's currency and FETCH the first SUPPLY record belonging to PART.

Figure 7.5. Split Screen After FETCH CURRENT PART RETAINING PART_SUPPLY



ZK-6072-GE

Now you can retest the MODIFY PART_STATUS.

```
dbq> MODIFY PART_STATUS
PART_STATUS [CHARACTER(1)]= X
dbq>
```

The DBQ prompt indicates the MODIFY was successful.

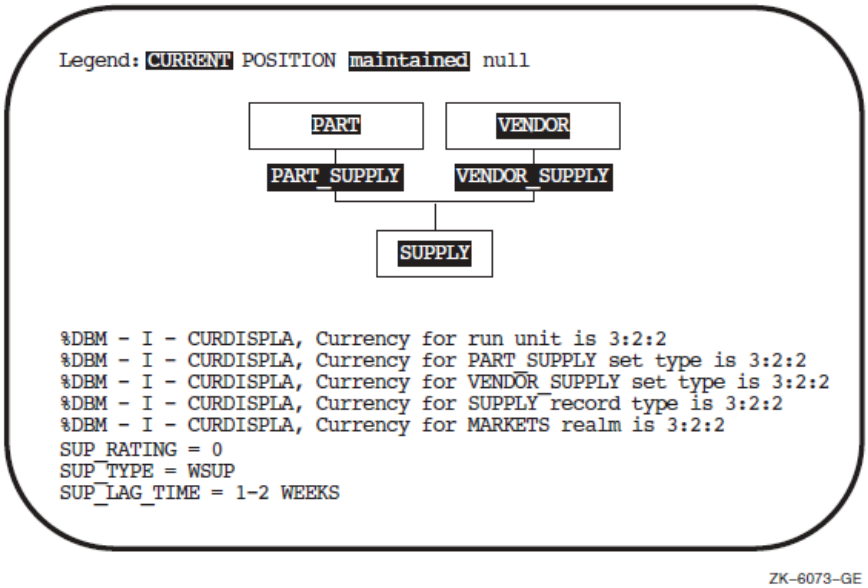
With the logic error found and fixed, you can test to see if the next execution of the FETCH NEXT WITHIN PART_SUPPLY fetches the next SUPPLY record belonging to the first PART record.

```
dbq> FETCH NEXT WITHIN PART_SUPPLY
```

The database keys displayed by the currency update messages in Figure 7.6 and Figure 7.7 are the same, thereby showing the A100-LOOP paragraph will fetch the next SUPPLY record owned by the first PART record.

Notice the data items also have the same value. Comparing data item contents instead of database key values is not a good practice because duplicate records may be allowed. For example, a PART may have two or more SUPPLY records containing the same data. Also, each SUPPLY record could point to a different owner in the VENDOR_SUPPLY set type.

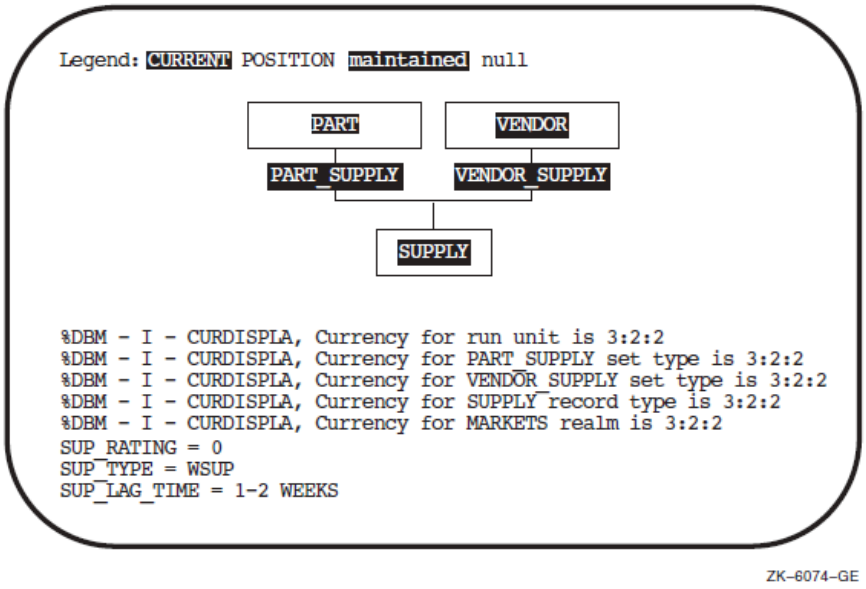
Figure 7.6. Split Screen After FETCH NEXT WITHIN PART_SUPPLY



To show that the record is indeed the second SUPPLY record belonging to the first PART record, execute the following statement:

```
dbq> FETCH 2 WITHIN PART_SUPPLY
```

Figure 7.7. Split Screen After FETCH 2 WITHIN PART_SUPPLY



7.3. Program Map Listings on Alpha or VAX

Listings are different on OpenVMS Alpha and OpenVMS I64 systems than they are on OpenVMS VAX systems. This section shows two listings on Alpha and I64 and two on VAX.

7.3.1. Listings on Alpha and I64

This section shows two compiler listing examples for OpenVMS Alpha and OpenVMS I64.

PARTSS1 Program Map Listing (Alpha, I64)

PARTSS1-PROGRAM in Example 7.3 includes the Oracle CODASYL DBMS data-names of the PARTSS1 subschema. The complete subschema can be obtained from the Oracle CDD/Repository DICTIONARY utility, using the following commands (refer to the Oracle CDD/Repository documentation):

```
$  DICTIONARY OPERATOR
CDO> SET OUTPUT filename.extension
CDO> SHO GENERIC CDD$DATABASE/FULL database-name
```

(The logical CDD$DEFAULT must have been previously defined.)

Example 7.3. PARTSS1-PROGRAM Compiler Listing (Alpha, I64)

```
PARTSS1-PROGRAM          Source Listing          18-JUN-2004 08:20:37  HP
 COBOL V2.8                 Page 1
0                        Source Listing          18-JUN-2004 08:17:19
 DEVICE:[COBOL.EXAMPLES]PARTSS1.COB;1

       1 IDENTIFICATION DIVISION.
       2 PROGRAM-ID.  PARTSS1-PROGRAM.
       3
       4 DATA DIVISION.
       5 SUB-SCHEMA SECTION.
       6 DB      PARTSS5 WITHIN PARTS FOR "DBM$IVP_OUTPUT:DBMPARTS".
       7
       8 PROCEDURE DIVISION.
       9 END PROGRAM PARTSS1-PROGRAM.

PARTSS1-PROGRAM          Source Listing          18-JUN-2004 08:20:37  HP
 COBOL V2.8                 Page 2
0                        Program Section Summary 18-JUN-2004 08:17:19
 DEVICE:[COBOL.EXAMPLES]PARTSS1.COB;1



PROGRAM SECTION INDEX

   Index  Name                Bytes     Alignment  Attributes
   -----  ----------------  ----------  ---------
 ---------------------------------------------------------------
      11  DBM$UWA_B                576  OCTA    4    PIC    OVR    REL
 GBL    SHR  NOEXE     RD    WRT  NOVEC
      12  DBM$SSC_B                 48  OCTA    4    PIC    CON    REL
 GBL  NOSHR  NOEXE     RD  NOWRT  NOVEC



DIAGNOSTICS SUMMARY

       Informationals       1 (suppressed)
       ----------------------
       Total                1

PARTSS1_PROGRAM\PARTSS1_PROGRAM Source Listing   18-JUN-2004 08:20:37  HP
 COBOL V2.8                 Page 3
```

```
0                Data Names in Alphabetic Order  18-JUN-2004 08:17:19
 DEVICE:[COBOL.EXAMPLES]PARTSS1.COB;1


Line    Level   Name              Location         Size        Bytes
 Usage    Category   Subs   Attribute
-----   -----   -------------   -------------   ----------   ----------
 --------  --------   ----   ---------
    6     01    CATEGORY         11  000000C8           23           23
 DISPLAY   Group               Glo
    6     02    CLASS_CODE       11  000000C8            2            2
 DISPLAY   AN                  Glo
    6     02    CLASS_DESC       11  000000CA           20           20
 DISPLAY   AN                  Glo
    6     02    CLASS_STATUS     11  000000DE            1            1
 DISPLAY   AN                  Glo
    6     02    COMP_MEASURE     11  000000F0            1            1
 DISPLAY   AN                  Glo
    6     02    COMP_OWNER_PART  11  000000E8            8            8
 DISPLAY   AN                  Glo
    6     02    COMP_QUANTITY    11  000000F1            5            3
 COMP-3    N                   Glo
    6     02    COMP_SUB_PART    11  000000E0            8            8
 DISPLAY   AN                  Glo
    6     01    COMPONENT        11  000000E0           20           20
 DISPLAY   Group               Glo
    6     01    DB-CONDITION     11  0000003C            9            4
 COMP      N                   Glo
    6     01    DB-CURRENT-RECORD-ID
                                 11  00000000            4            2
 COMP      N                   Glo
    6     01    DB-CURRENT-RECORD-NAME
                                 11  00000019           31           31
 DISPLAY   AN                  Glo
    6     01    DB-KEY           11  0000007A           18            8
 COMP      N                   Glo
    6     01    DB-UWA           11  00000000          108          108
 DISPLAY   AN                  Glo
    6     02    EMP_FIRST_NAME   11  0000010F           10           10
 DISPLAY   AN                  Glo
    6     02    EMP_ID           11  000000F8            5            3
 COMP-3    N                   Glo
    6     02    EMP_LAST_NAME    11  000000FB           20           20
 DISPLAY   AN                  Glo
    6     02    EMP_LOC          11  00000120            5            5
 DISPLAY   AN                  Glo
    6     02    EMP_PHONE        11  00000119            7            7
 DISPLAY   AN                  Glo
    6     01    EMPLOYEE         11  000000F8           45           45
 DISPLAY   Group               Glo
    6     02    GROUP_NAME       11  00000128           20           20
 DISPLAY   AN                  Glo
    6     01    PART             11  00000140           71           71
 DISPLAY   Group               Glo
    6     02    PART_COST        11  00000180            9            5
 COMP-3    N                   Glo
    6     02    PART_DESC        11  00000148           50           50
 DISPLAY   AN                  Glo
```

```
    6     02    PART_ID          11  00000140            8            8
 DISPLAY   AN                  Glo
    6     02    PART_PRICE       11  0000017B            9            5
 COMP-3    N                   Glo
    6     02    PART_STATUS      11  0000017A            1            1
 DISPLAY   AN                  Glo
    6     02    PART_SUPPORT     11  00000185            2            2
 DISPLAY   AN                  Glo
    6     01    PR_QUOTE         11  00000188           26           26
 DISPLAY   Group               Glo
    6     02    QUOTE_DATE       11  0000018F            6            6
 DISPLAY   AN                  Glo
    6     02    QUOTE_ID         11  00000188            7            7
 DISPLAY   AN                  Glo
    6     02    QUOTE_MIN_ORDER  11  00000195            5            3
 COMP-3    N                   Glo
    6     02    QUOTE_QTY_PRICE  11  0000019D            9            5
 COMP-3    N                   Glo
    6     02    QUOTE_UNIT_PRIC  11  00000198            9            5
 COMP-3    N                   Glo
    6     02    SUP_LAG_TIME     11  000001AD           10           10
 DISPLAY   AN                  Glo
    6     02    SUP_RATING       11  000001A8            1            1
 DISPLAY   AN                  Glo
    6     02    SUP_TYPE         11  000001A9            4            4
 DISPLAY   AN                  Glo
    6     01    SUPPLY           11  000001A8           15           15
 DISPLAY   Group               Glo
    6     02    VEND_ADDRESS     11  00000206           15           15
 DISPLAY   AN            1     Glo
    6     02    VEND_CONTACT     11  000001E8           30           30
 DISPLAY   AN                  Glo
    6     02    VEND_ID          11  000001B8            8            8
 DISPLAY   AN                  Glo
    6     02    VEND_NAME        11  000001C0           40           40
 DISPLAY   AN                  Glo
    6     02    VEND_PHONE       11  00000233           10           10
 DISPLAY   AN                  Glo
    6     01    VENDOR           11  000001B8          133          133
 DISPLAY   Group               Glo
    6     01    WK_GROUP         11  00000128           20           20
 DISPLAY   Group               Glo

PARTSS1_PROGRAM\PARTSS1_PROGRAM Source Listing       18-JUN-2004 08:20:37
 HP COBOL                      Page 4
0                    Procedure Names in Alpha Order  18-JUN-2004 08:17:19
 DEVICE:[COBOL.EXAMPLES]PARTSS1.COB;1


Line    Name                                 Location      Type
-----   ------------------------------   -------------
 ----------------------------------------------------
    2   PARTSS1-PROGRAM                            **      Program

PARTSS1_PROGRAM\PARTSS1_PROGRAM Source Listing       18-JUN-2004 08:20:37
 HP COBOL V2.8                 Page 5
0                              Compilation Summary  18-JUN-2004 08:17:19
 DEVICE:[COBOL.EXAMPLES]PARTSS1.COB;1
```

```
COMMAND QUALIFIERS

   COBOL

      /NOALIGNMENT                                                    /
GRANULARITY = QUAD
      /NOANALYSIS_DATA                                                /
NOINCLUDE
      /NOANSI_FORMAT                                                  /LIST
      /ARCHITECTURE = GENERIC                                         /
NOMACHINE_CODE
      /ARITHMETIC = NATIVE                                            /MAP =
 ALPHABETICAL
      /NOAUDIT                                                        /
MATH_INTERMEDIATE = FLOAT
      /CHECK = (NOPERFORM, NOBOUNDS, NODECIMAL, NODUPLICATE_KEYS)     /
NATIONALITY = US
      /NOCONDITIONALS                                                 /
NOOBJECT
      /NOCONVERT = LEADING_BLANKS                                     /
OPTIMIZE = (LEVEL=4,TUNE=GENERIC)
      /NOCOPY_LIST                                                    /
RESERVED_WORDS = (XOPEN,

 NOFOREIGN_EXTENSIONS, NO200X)
      /NOCROSS_REFERENCE                                              /
NOSEPARATE_COMPILATION
      /DEBUG = (NOSYMBOLS, TRACEBACK)                                 /
NOSEQUENCE_CHECK
      /NODEPENDENCY_DATA                                              /
STANDARD = (NOXOPEN, NOSYNTAX,

 NOV3, 85, NOMIA)
      /NODIAGNOSTICS                                                  /NOTIE
      /NODISPLAY_FORMATTED                                            /
NOTRUNCATE
      /NOFIPS                                                         /VFC
      /NOFLAGGER                                                      /
WARNINGS = (NOINFORMATION, OTHER)
      /FLOAT = D_FLOAT


COMPILATION STATISTICS

  CPU time:          1.59 seconds
  Elapsed time:      7.59 seconds
  Pagefaults:        1014
  I/O Count:          343
  Source lines:         9

  339 lines per CPU minute.
```

PARTSS3 Program Map Listing (Alpha, I64)

PARTSS3-PROGRAM in Example 7.4 includes the Oracle CODASYL DBMS data-names of the PARTSS3 subschema.

Example 7.4. PARTSS3-PROGRAM Compiler Listing (Alpha, I64)

```
PARTSS3-PROGRAM          Source Listing          18-JUN-2004 08:33:40  HP
 COBOL V2.8                 Page 1
0                        Source Listing          18-JUN-2004 08:30:39
 DEVICE:[COBOL.EXAMPLES]PARTSSE.COB;1

       1 IDENTIFICATION DIVISION.
       2 PROGRAM-ID.  PARTSS3-PROGRAM.
       3
       4 DATA DIVISION.
       5 SUB-SCHEMA SECTION.
       6 DB      PARTSS3 WITHIN PARTS FOR "DBM$IVP_OUTPUT:DBMPARTS".
       7
       8 PROCEDURE DIVISION.
       9 END PROGRAM PARTSS3-PROGRAM.


PARTSS3-PROGRAM          Source Listing          18-JUN-2004 08:33:40  HP
 COBOL V2.8                 Page 2
0                        Program Section Summary 18-JUN-2004 08:30:39
 DEVICE:[COBOL.EXAMPLES]PARTSSE.COB;1



PROGRAM SECTION INDEX

   Index  Name          Bytes     Alignment  Attributes
   -----  ----------- ----------  ---------
 ---------------------------------------------------------------
      11  DBM$UWA_B          376  OCTA    4    PIC    OVR    REL    GBL
 SHR  NOEXE     RD    WRT  NOVEC
      12  DBM$SSC_B           48  OCTA    4    PIC    CON    REL    GBL
 NOSHR  NOEXE     RD  NOWRT  NOVEC



DIAGNOSTICS SUMMARY

       Informationals       1 (suppressed)
       ----------------------
       Total                1


PARTSS3_PROGRAM\PARTSS3_PROGRAM Source Listing   18-JUN-2004 08:33:40  HP
 COBOL V2.8                 Page 3
0                Data Names in Alphabetic Order  18-JUN-2004 08:30:39
 DEVICE:[COBOL.EXAMPLES]PARTSSE.COB;1


Line    Level   Name               Location         Size        Bytes
 Usage    Category   Subs   Attribute
```

```
-----   -----   --------------   -------------   ----------   ----------
 --------  --------   ----   ---------
    6     01    DB-CONDITION      11  0000003C            9            4
 COMP      N                   Glo
    6     01    DB-CURRENT-RECORD-ID
                                 11  00000000            4            2
 COMP      N                   Glo
    6     01    DB-CURRENT-RECORD-NAME
                                 11  00000019           31           31
 DISPLAY   AN                  Glo
    6     01    DB-KEY            11  0000007A           18            8
 COMP      N                   Glo
    6     01    DB-UWA            11  00000000          108          108
 DISPLAY   AN                  Glo
    6     01    PART              11  000000A0           61           61
 DISPLAY   Group               Glo
    6     02    PART_DESC         11  000000A8           50           50
 DISPLAY   AN                  Glo
    6     02    PART_ID           11  000000A0            8            8
 DISPLAY   AN                  Glo
    6     02    PART_STATUS       11  000000DA            1            1
 DISPLAY   AN                  Glo
    6     02    PART_SUPPORT      11  000000DB            2            2
 DISPLAY   AN                  Glo
    6     02    SUP_LAG_TIME      11  000000E5           10           10
 DISPLAY   AN                  Glo
    6     02    SUP_RATING        11  000000E0            1            1
 DISPLAY   AN                  Glo
    6     02    SUP_TYPE          11  000000E1            4            4
 DISPLAY   AN                  Glo
    6     01    SUPPLY            11  000000E0           15           15
 DISPLAY   Group               Glo
    6     02    VEND_ADDRESS      11  0000013E           15           15
 DISPLAY   AN            1     Glo
    6     02    VEND_CONTACT      11  00000120           30           30
 DISPLAY   AN                  Glo
    6     02    VEND_ID           11  000000F0            8            8
 DISPLAY   AN                  Glo
    6     02    VEND_NAME         11  000000F8           40           40
 DISPLAY   AN                  Glo
    6     02    VEND_PHONE        11  0000016B           10           10
 DISPLAY   N                   Glo
    6     01    VENDOR            11  000000F0          133          133
 DISPLAY   Group               Glo

PARTSS3_PROGRAM\PARTSS3_PROGRAM Source Listing     18-JUN-2004 08:33:40  HP
 COBOL V2.8                 Page 4
0                  Procedure Names in Alpha Order  18-JUN-2004 08:30:39
 DEVICE:[COBOL.EXAMPLES]PARTSSE.COB;1


Line    Name           Location      Type
-----   ----------   -------------
 ---------------------------------------------------------------------------
    2   PARTSS3-PROGRAM      **      Program

PARTSS3_PROGRAM\PARTSS3_PROGRAM Source Listing      18-JUN-2004 08:33:40
 HP COBOL V2.8                 Page 5
```

```
0                           Compilation Summary     18-JUN-2004 08:30:39
 DEVICE:[COBOL.EXAMPLES]PARTSSE.COB;1



COMMAND QUALIFIERS

   COBOL

      /NOALIGNMENT                                                    /
GRANULARITY = QUAD
      /NOANALYSIS_DATA                                                /
NOINCLUDE
      /NOANSI_FORMAT                                                  /LIST
      /ARCHITECTURE = GENERIC                                         /
NOMACHINE_CODE
      /ARITHMETIC = NATIVE                                            /MAP =
 ALPHABETICAL
      /NOAUDIT                                                        /
MATH_INTERMEDIATE = FLOAT
      /CHECK = (NOPERFORM, NOBOUNDS, NODECIMAL, NODUPLICATE_KEYS)     /
NATIONALITY = US
      /NOCONDITIONALS                                                 /
NOOBJECT
      /NOCONVERT = LEADING_BLANKS                                     /
OPTIMIZE = (LEVEL=4,TUNE=GENERIC)
      /NOCOPY_LIST                                                    /
RESERVED_WORDS = (XOPEN,

 NOFOREIGN_EXTENSIONS, NO200X)
      /NOCROSS_REFERENCE                                              /
NOSEPARATE_COMPILATION
      /DEBUG = (NOSYMBOLS, TRACEBACK)                                 /
NOSEQUENCE_CHECK
      /NODEPENDENCY_DATA                                              /
STANDARD = (NOXOPEN, NOSYNTAX,

 NOV3, 85, NOMIA)
      /NODIAGNOSTICS                                                  /NOTIE
      /NODISPLAY_FORMATTED                                            /
NOTRUNCATE
      /NOFIPS                                                         /VFC
      /NOFLAGGER                                                      /
WARNINGS = (NOINFORMATION, OTHER)
      /FLOAT = D_FLOAT


COMPILATION STATISTICS

  CPU time:          1.59 seconds
  Elapsed time:      7.63 seconds
  Pagefaults:        1053
  I/O Count:          340
  Source lines:         9

  339 lines per CPU minute.    <>
```

7.3.2. Listings on VAX

This section shows two compiler listing examples on OpenVMS VAX.

PARTSS1 Program Map Listing (VAX)

The VSI COBOL for OpenVMS VAX (formerly Compaq COBOL) compiler produces listings that are different in some respects from those produced by VSI COBOL for OpenVMS Alpha. Following are examples of VAX listings.

PARTSS1-PROGRAM in Example 7.5 includes the VSI COBOL for OpenVMS VAX subschema map of the PARTSS1 subschema.

Example 7.5. PARTSS1-PROGRAM Compiler Listing (VAX)

```
PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   1
Source Listing                  31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)

    1         IDENTIFICATION DIVISION.
    2         PROGRAM-ID. PARTSS1-PROGRAM.
    3
    4         DATA DIVISION.
    5         SUB-SCHEMA SECTION.
    6         DB PARTSS5 WITHIN PARTS.
    7
    8         PROCEDURE DIVISION.
    9         END PROGRAM PARTSS1-PROGRAM.
PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   2
Data Names in Alphabetic Order  31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)

 Line   Level   Name              Location         Size        Bytes
 Usage   Category   Subs   Attribute

    6      01   CATEGORY       7  000000AC           23           23
 DISPLAY   Group               Glo
    6      02   CLASS_CODE     7  000000AC            2            2
 DISPLAY   AN                  Glo
    6      02   CLASS_DESC     7  000000AE           20           20
 DISPLAY   AN                  Glo
    6      02   CLASS_STATUS   7  000000C2            1            1
 DISPLAY   AN                  Glo
    6      02   COMP_MEASURE   7  000000D4            1            1
 DISPLAY   AN                  Glo
    6      02   COMP_OWNER_PART
                               7  000000CC            8            8
 DISPLAY   AN                  Glo
    6      02   COMP_QUANTITY  7  000000D5            5            3
 COMP-3    N                   Glo
    6      02   COMP_SUB_PART  7  000000C4            8            8
 DISPLAY   AN                  Glo
    6      01   COMPONENT      7  000000C4           20           20
 DISPLAY   Group               Glo
```

```
   6      01   DB-CONDITION   7  00000028            9            4   COMP
     N                   Glo
   6      01   DB-CURRENT-RECORD-ID
                             7  00000000            4            2   COMP
     N                   Glo
   6      01   DB-CURRENT-RECORD-NAME
                             7  00000005           31           31
DISPLAY   AN                 Glo
   6      01   DB-KEY         7  00000064           18            8   COMP
     N                   Glo
   6      01   DB-UWA         7  00000000          108          108
DISPLAY   AN                 Glo
   6      02   EMP_FIRST_NAME 7  000000EF           10           10
DISPLAY   AN                 Glo
   6      02   EMP_ID         7  000000D8            5            3
COMP-3    N                  Glo
   6      02   EMP_LAST_NAME  7  000000DB           20           20
DISPLAY   AN                 Glo
   6      02   EMP_LOC        7  00000100            5            5
DISPLAY   AN                 Glo
   6      02   EMP_PHONE      7  000000F9            7            7
DISPLAY   AN                 Glo
   6      01   EMPLOYEE       7  000000D8           45           45
DISPLAY   Group              Glo
   6      02   GROUP_NAME     7  00000108           20           20
DISPLAY   AN                 Glo
   6      01   PART           7  0000011C           71           71
DISPLAY   Group              Glo
   6      02   PART_COST      7  0000015C            9            5
COMP-3    N                  Glo
   6      02   PART_DESC      7  00000124           50           50
DISPLAY   AN                 Glo
   6      02   PART_ID        7  0000011C            8            8
DISPLAY   AN                 Glo
   6      02   PART_PRICE     7  00000157            9            5
COMP-3    N                  Glo
   6      02   PART_STATUS    7  00000156            1            1
DISPLAY   AN                 Glo
   6      02   PART_SUPPORT   7  00000161            2            2
DISPLAY   AN                 Glo
   6      01   PR_QUOTE       7  00000164           26           26
DISPLAY   Group              Glo
   6      02   QUOTE_DATE     7  0000016B            6            6
DISPLAY   AN                 Glo
   6      02   QUOTE_ID       7  00000164            7            7
DISPLAY   AN                 Glo
   6      02   QUOTE_MIN_ORDER7  00000171            5            3
COMP-3    N                  Glo
   6      02   QUOTE_QTY_PRICE7  00000179            9            5
COMP-3    N                  Glo
   6      02   QUOTE_UNIT_PRIC7  00000174            9            5
COMP-3    N                  Glo
   6      02   SUP_LAG_TIME   7  00000185           10           10
DISPLAY   AN                 Glo
   6      02   SUP_RATING     7  00000180            1            1
DISPLAY   AN                 Glo
   6      02   SUP_TYPE       7  00000181            4            4
DISPLAY   AN                 Glo
```

```
    6      01   SUPPLY         7  00000180          15          15
 DISPLAY   Group               Glo
    6      02   VEND_ADDRESS   7  000001DE          15          15
 DISPLAY   AN            1     Glo
    6      02   VEND_CONTACT   7  000001C0          30          30
 DISPLAY   AN                  Glo
    6      02   VEND_ID        7  00000190           8           8
 DISPLAY   AN                  Glo
    6      02   VEND_NAME      7  00000198          40          40
 DISPLAY   AN                  Glo
    6      02   VEND_PHONE     7  0000020B          10          10
 DISPLAY   AN                  Glo
    6      01   VENDOR         7  00000190         133         133
 DISPLAY   Group               Glo
    6      01   WK_GROUP       7  00000108          20          20
 DISPLAY   Group               Glo
PARTSS1-PROGRAM                      31-May-2004 14:08:50  Compaq COBOL
 V5.7-63          Page   3
Procedure Names in Alphabetic Order  31-May-2004 14:03:05
 [SYSTEST.DBM]PARTSS1-PROGRAM.COB;3 (1)

 Line   Name                                   Location   Type

    2   PARTSS1-PROGRAM                     0  00000000   Program
PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   4
References                      31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)

DBM$_NOT_BOUND

PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   5
Sub-schema Map                  31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)
* SYS$COMMON:[SYSTEST.DBM.CDDPLUS1]PARTS.DBM$SUBSCHEMAS.PARTSS5
*
* Subschema version number:  31-MAY-2004 14:06:24.23
*
SUBSCHEMA NAME PARTSS5 FOR CDDPLUS1]PARTS SCHEMA
REALM BUY

REALM MAKE

REALM MARKET
REALM PERSONNEL

* Within areas:    BUY
*                  MAKE
* Owner of sets:   CATEGORY_PART
* Member of sets:  ALL_CATEGORIES
*
01  CATEGORY.
    02  CLASS_CODE          PIC X(2).
    02  CLASS_DESC          PIC X(20).
    02  CLASS_STATUS        PIC X.
* Within areas:    MAKE
* Member of sets:  PART_USES
```

```
*                  PART_USED_ON
*
01  COMPONENT.
    02  COMP_SUB_PART       PIC X(8).
    02  COMP_OWNER_PART     PIC X(8).
    02  COMP_MEASURE        PIC X.
    02  COMP_QUANTITY       PIC S9(3)V9(2) COMP-3.
* Within areas:    PERSONNEL
* Owner of sets:   MANAGES
*                  RESPONSIBLE_FOR
* Member of sets:  ALL_EMPLOYEES
*                  CONSISTS_OF
*
01  EMPLOYEE.
    02  EMP_ID              PIC S9(5) COMP-3.
    02  EMP_LAST_NAME       PIC X(20).
    02  EMP_FIRST_NAME      PIC X(10).
    02  EMP_PHONE           PIC X(7).
    02  EMP_LOC             PIC X(5).
* Within areas:    PERSONNEL
* Owner of sets:   CONSISTS_OF
* Member of sets:  MANAGES
*
01  WK_GROUP.
    02  GROUP_NAME          PIC X(20).
* Within areas:    BUY
*                  MAKE
* Owner of sets:   PART_USES
PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   6
Sub-schema Map                  31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)

*                  PART_INFO
*                  PART_USED_ON
* Member of sets:  ALL_PARTS
*                  ALL_PARTS_ACTIVE
*                  CATEGORY_PART
*                  RESPONSIBLE_FOR
*
01  PART.
    02  PART_ID             PIC X(8).
    02  PART_DESC           PIC X(50).
    02  PART_STATUS         PIC X.
    02  PART_PRICE          PIC S9(6)V9(3) COMP-3.
    02  PART_COST           PIC S9(6)V9(3) COMP-3.
    02  PART_SUPPORT        PIC X(2).
* Within areas:    MARKET
* Member of sets:  PART_INFO
*
01  PR_QUOTE.
    02  QUOTE_ID            PIC X(7).
    02  QUOTE_DATE          PIC X(6).
    02  QUOTE_MIN_ORDER     PIC S9(5) COMP-3.
    02  QUOTE_UNIT_PRIC     PIC S9(6)V9(3) COMP-3.
    02  QUOTE_QTY_PRICE     PIC S9(6)V9(3) COMP-3.
* Within areas:    MARKET
* Member of sets:  PART_INFO
```

```
*                  VENDOR_SUPPLY
*
01  SUPPLY.
    02  SUP_RATING          PIC X.
    02  SUP_TYPE            PIC X(4).
    02  SUP_LAG_TIME        PIC X(10).
* Within areas:    MARKET
* Owner of sets:   VENDOR_SUPPLY
* Member of sets:  ALL_VENDORS
*
01  VENDOR.
    02  VEND_ID             PIC X(8).
    02  VEND_NAME           PIC X(40).
    02  VEND_CONTACT        PIC X(30).
    02  VEND_ADDRESS        PIC X(15) OCCURS 3 TIMES.
    02  VEND_PHONE          PIC X(10).
SET NAME ALL_CATEGORIES
    OWNER SYSTEM
    MEMBER CATEGORY
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER SYSTEM DEFAULT
SET NAME ALL_EMPLOYEES
    OWNER SYSTEM
    MEMBER EMPLOYEE
        INSERTION AUTOMATIC
        RETENTION FIXED
PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   7
Sub-schema Map                  31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)

        ORDER SYSTEM DEFAULT
SET NAME ALL_PARTS
    OWNER SYSTEM
    MEMBER PART
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER SYSTEM DEFAULT
SET NAME ALL_PARTS_ACTIVE
    OWNER SYSTEM
    MEMBER PART
        INSERTION AUTOMATIC
        RETENTION OPTIONAL
        ORDER SYSTEM DEFAULT
SET NAME ALL_VENDORS
    OWNER SYSTEM
    MEMBER VENDOR
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER SORTED
SET NAME CATEGORY_PART
    OWNER CATEGORY
    MEMBER PART
        INSERTION AUTOMATIC
        RETENTION MANDATORY
        ORDER SORTED
SET NAME CONSISTS_OF
```

```
    OWNER WK_GROUP
    MEMBER EMPLOYEE
        INSERTION MANUAL
        RETENTION OPTIONAL
        ORDER SORTED
SET NAME MANAGES
    OWNER EMPLOYEE
    MEMBER WK_GROUP
        INSERTION AUTOMATIC
        RETENTION OPTIONAL
        ORDER NEXT
SET NAME PART_USES
    OWNER PART
    MEMBER COMPONENT
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER NEXT
SET NAME PART_INFO
    OWNER PART
    MEMBER PR_QUOTE
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER NEXT
PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   8
Sub-schema Map                  31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)

    MEMBER SUPPLY
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER NEXT
SET NAME PART_USED_ON
    OWNER PART
    MEMBER COMPONENT
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER NEXT
SET NAME RESPONSIBLE_FOR
    OWNER EMPLOYEE
    MEMBER PART
        INSERTION MANUAL
        RETENTION OPTIONAL
        ORDER NEXT
SET NAME VENDOR_SUPPLY
    OWNER VENDOR
    MEMBER SUPPLY
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER NEXT

PARTSS1-PROGRAM                 31-May-2004 14:08:50  Compaq COBOL V5.7-63
         Page   9
Compilation Summary             31-May-2004 14:03:05  [SYSTEST.DBM]PARTSS1-
PROGRAM.COB;3 (1)
PROGRAM SECTIONS

    Name                        Bytes   Attributes
```

```
  0 $CODE                          6     PIC   CON   REL   LCL   SHR   EXE
   RD NOWRT Align(2)
  3 COB$NAMES_____2               24     PIC   CON   REL   LCL   SHR NOEXE
   RD NOWRT Align(2)
  4 COB$NAMES_____4               16     PIC   CON   REL   LCL   SHR NOEXE
   RD NOWRT Align(2)
  5 DBM$SSC_B                     28     PIC   CON   REL   GBL NOSHR NOEXE
   RD NOWRT Align(2)
  7 DBM$UWA_B                    533     PIC   OVR   REL   GBL   SHR NOEXE
   RD   WRT Align(2)

DIAGNOSTICS

    Informational:      1 (suppressed by command qualifier)

COMMAND QUALIFIERS

    COBOL /LIST/MAP PARTSS1-PROGRAM.COB

    /NOCOPY_LIST  /NOMACHINE_CODE  /NOCROSS_REFERENCE
    /NOANSI_FORMAT  /NOSEQUENCE_CHECK  /MAP=ALPHABETICAL
    /NOTRUNCATE  /NOAUDIT  /NOCONDITIONALS
    /CHECK=(NOPERFORM,NOBOUNDS,NODUPLICATE_KEYS)  /
DEBUG=(NOSYMBOLS,TRACEBACK)
    /WARNINGS=(NOSTANDARD,OTHER,NOINFORMATION)  /NODEPENDENCY_DATA
    /STANDARD=(NOSYNTAX,NOPDP11,NOV3,85,NOALPHA_AXP)  /NOFIPS
    /LIST  /OBJECT /NODIAGNOSTICS /NOFLAGGER /NOANALYSIS_DATA
    /INSTRUCTION_SET=DECIMAL_STRING /DESIGN=(NOPLACEHOLDERS,NOCOMMENTS)
    /NATIONALITY=US

STATISTICS

    Run Time:           2.16 seconds
    Elapsed Time:       5.29 seconds
    Page Faults:        14236
    Dynamic Memory:     9695 pages
```

PARTSS3 Program Map Listing (VAX)

PARTSS3-PROGRAM in Example 7.6 includes the VSI COBOL for OpenVMS VAX subschema map of the PARTSS3 subschema.

Example 7.6. PARTSS3-PROGRAM Compiler Listing (VAX)

```
PARTSS3-PROGRAM                     31-May-2004 12:31:18  Compaq COBOL
 V5.7-63          Page   1
Source Listing                      31-May-2004 12:25:37
 [SYSTEST.DBM]PARTSS3-PROGRAM.COB;2 (1)

    1         IDENTIFICATION DIVISION.
    2         PROGRAM-ID. PARTSS3-PROGRAM.
    3
    4         DATA DIVISION.
    5         SUB-SCHEMA SECTION.
    6         DB PARTSS3 WITHIN PARTS.
    7
    8         PROCEDURE DIVISION.
```

```
    9         END PROGRAM PARTSS3-PROGRAM.
PARTSS3-PROGRAM                     31-May-2004 12:31:18  Compaq COBOL
 V5.7-63          Page   2
Data Names in Alphabetic Order      31-May-2004 12:25:37
 [SYSTEST.DBM]PARTSS3-PROGRAM.COB;2 (1)

 Line   Level   Name                    Location       Size        Bytes
  Usage     Category   Subs   Attribute

    6      01   DB-CONDITION         7  00000028            9            4
  COMP      N                  Glo
    6      01   DB-CURRENT-RECORD-ID
                                     7  00000000            4            2
  COMP      N                  Glo
    6      01   DB-CURRENT-RECORD-NAME
                                     7  00000005           31           31
  DISPLAY   AN                 Glo
    6      01   DB-KEY               7  00000064           18            8
  COMP      N                  Glo
    6      01   DB-UWA               7  00000000          108          108
  DISPLAY   AN                 Glo
    6      01   PART                 7  00000084           61           61
  DISPLAY   Group              Glo
    6      02   PART_DESC            7  0000008C           50           50
  DISPLAY   AN                 Glo
    6      02   PART_ID              7  00000084            8            8
  DISPLAY   AN                 Glo
    6      02   PART_STATUS          7  000000BE            1            1
  DISPLAY   AN                 Glo
    6      02   PART_SUPPORT         7  000000BF            2            2
  DISPLAY   AN                 Glo
    6      02   SUP_LAG_TIME         7  000000C9           10           10
  DISPLAY   AN                 Glo
    6      02   SUP_RATING           7  000000C4            1            1
  DISPLAY   AN                 Glo
    6      02   SUP_TYPE             7  000000C5            4            4
  DISPLAY   AN                 Glo
    6      01   SUPPLY               7  000000C4           15           15
  DISPLAY   Group              Glo
    6      02   VEND_ADDRESS         7  00000122           15           15
  DISPLAY   AN            1    Glo
    6      02   VEND_CONTACT         7  00000104           30           30
  DISPLAY   AN                 Glo
    6      02   VEND_ID              7  000000D4            8            8
  DISPLAY   AN                 Glo
    6      02   VEND_NAME            7  000000DC           40           40
  DISPLAY   AN                 Glo
    6      02   VEND_PHONE           7  0000014F           10           10
  DISPLAY   N                  Glo
    6      01   VENDOR               7  000000D4          133          133
  DISPLAY   Group              Glo
PARTSS3-PROGRAM                     31-May-2004 12:31:18    Compaq COBOL
 V5.7-63          Page   3
Procedure Names in Alphabetic Order 31-May-2004 12:25:37
 [SYSTEST.DBM]PARTSS3-PROGRAM.COB;2 (1)

 Line   Name                                  Location   Type
```

```
    2   PARTSS3-PROGRAM                     0  00000000   Program
PARTSS3-PROGRAM                     31-May-2004 12:31:18 Compaq COBOL
 V5.7-63          Page   4
External References                 31-May-2004 12:25:37
 [SYSTEST.DBM]PARTSS3-PROGRAM.COB;2 (1)

DBM$_NOT_BOUND

PARTSS3-PROGRAM                     31-May-2004 12:31:18  Compaq COBOL
 V5.7-63          Page   5
Sub-schema Map                      31-May-2004 12:25:37
 [SYSTEST.DBM]PARTSS3-PROGRAM.COB;2 (1)
* SYS$COMMON:[SYSTEST.DBM.CDDPLUS1]PARTS.DBM$SUBSCHEMAS.PARTSS3
*
* Subschema version number:  31-MAY-2004 12:28:53.22
*
SUBSCHEMA NAME PARTSS3 FOR CDDPLUS1]PARTS SCHEMA
REALM MARKETS

* Within areas:    MARKETS
* Owner of sets:   PART_SUPPLY
*
01  PART.
    02  PART_ID             PIC X(8).
    02  PART_DESC           PIC X(50).
    02  PART_STATUS         PIC X.
    02  PART_SUPPORT        PIC X(2).
* Within areas:    MARKETS
* Member of sets:  PART_SUPPLY
*                  VENDOR_SUPPLY
*
01  SUPPLY.
    02  SUP_RATING          PIC X.
    02  SUP_TYPE            PIC X(4).
    02  SUP_LAG_TIME        PIC X(10).
* Within areas:    MARKETS
* Owner of sets:   VENDOR_SUPPLY
*
01  VENDOR.
    02  VEND_ID             PIC X(8).
    02  VEND_NAME           PIC X(40).
    02  VEND_CONTACT        PIC X(30).
    02  VEND_ADDRESS        PIC X(15) OCCURS 3 TIMES.
    02  VEND_PHONE          PIC 9(10).
SET NAME PART_SUPPLY
    OWNER PART
    MEMBER SUPPLY
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER NEXT
SET NAME VENDOR_SUPPLY
    OWNER VENDOR
    MEMBER SUPPLY
        INSERTION AUTOMATIC
        RETENTION FIXED
        ORDER NEXT
```

```
PARTSS3-PROGRAM                     31-May-2004 12:31:18  Compaq COBOL
 V5.7-63          Page   6
Compilation Summary                 31-May-2004 12:25:37
 [SYSTEST.DBM]PARTSS3-PROGRAM.COB;2 (1)

PROGRAM SECTIONS

  Name               Bytes   Attributes

0 $CODE               6     PIC   CON   REL   LCL   SHR   EXE   RD NOWRT
 Align(2)
3 COB$NAMES_____2    24     PIC   CON   REL   LCL   SHR NOEXE   RD NOWRT
 Align(2)
4 COB$NAMES_____4    16     PIC   CON   REL   LCL   SHR NOEXE   RD NOWRT
 Align(2)
5 DBM$SSC_B          28     PIC   CON   REL   GBL NOSHR NOEXE   RD NOWRT
 Align(2)
7 DBM$UWA_B         345     PIC   OVR   REL   GBL   SHR NOEXE   RD   WRT
 Align(2)

DIAGNOSTICS

    Informational:      1 (suppressed by command qualifier)

COMMAND QUALIFIERS

    COBOL /LIST/MAP PARTSS3-PROGRAM.COB

    /NOCOPY_LIST  /NOMACHINE_CODE  /NOCROSS_REFERENCE
    /NOANSI_FORMAT  /NOSEQUENCE_CHECK  /MAP=ALPHABETICAL
    /NOTRUNCATE  /NOAUDIT  /NOCONDITIONALS
    /CHECK=(NOPERFORM,NOBOUNDS,NODUPLICATE_KEYS)  /
DEBUG=(NOSYMBOLS,TRACEBACK)
    /WARNINGS=(NOSTANDARD,OTHER,NOINFORMATION)  /NODEPENDENCY_DATA
    /STANDARD=(NOSYNTAX,NOPDP11,NOV3,85,NOALPHA_AXP)  /NOFIPS
    /LIST  /OBJECT /NODIAGNOSTICS /NOFLAGGER /NOANALYSIS_DATA
    /INSTRUCTION_SET=DECIMAL_STRING /DESIGN=(NOPLACEHOLDERS,NOCOMMENTS)
    /NATIONALITY=US

STATISTICS

    Run Time:           1.76 seconds
    Elapsed Time:       4.23 seconds
    Page Faults:        13713
    Dynamic Memory:     8790 pages    <>
```

Chapter 8. Database Programming Examples

The next few pages show programming examples of how to do the following:

- Populate a database
- Back up a database
- Access and display database information
- Create new record relationships

This chapter also provides an example of how to create a bill of materials and sample runs of some of the programming examples.

8.1. Populating a Database

The DBMPARTLD program in Example 8.1 loads a series of sequential data files into the PARTS database. The PARTS database consists of a NEW root file with a default extension of .ROO describing the database instance and a series of .DBS storage files containing the actual data records. PARTS is the schema relative to the current position in CDD/Repository when the program is compiled. As the DBCS inserts the records, it creates set relationships based on the PARTSS1 subschema definitions. In the DB statement PARTS and NEW can be logical names. If PARTS is not a logical name, VSI COBOL for OpenVMS appends PARTS to CDD$DEFAULT; for example, CDD$DEFAULT.PARTS. If NEW is not a logical name, the DBCS appends .ROO as the default file type; for example, NEW.ROO.

Example 8.1. Populating a Database

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   DBMPARTLD.
*********************************************************
*                                                       *
* This program loads the PARTS database                 *
*                                                       *
*********************************************************
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT MAKE-FILE
                ASSIGN TO "DBM$PARTS:DBMMAKE.DAT".
        SELECT BUY-FILE
                ASSIGN TO "DBM$PARTS:DBMBUY.DAT".
        SELECT VENDOR-FILE
                ASSIGN TO "DBM$PARTS:DBMVENDOR.DAT".
        SELECT EMPLOYEE-FILE
                ASSIGN TO "DBM$PARTS:DBMEMPLOY.DAT".
        SELECT COMPONENT-FILE
                ASSIGN TO "DBM$PARTS:DBMCOMPON.DAT".
        SELECT SUPPLY-FILE
                ASSIGN TO "DBM$PARTS:DBMSUPPLY.DAT".
                SELECT DIVISION-FILE
                ASSIGN TO "DBM$PARTS:DBMSUPER.DAT".
```

```
        SELECT RESP-FOR-FILE
                ASSIGN TO "DBM$PARTS:DBMRESPON.DAT".
DATA DIVISION.
SUB-SCHEMA SECTION.
        DB PARTSS1 WITHIN PARTS FOR NEW.
FILE SECTION.
FD      MAKE-FILE
        RECORD VARYING FROM 24 TO 80 CHARACTERS.
01      MAKE-PART-RECORD.
            02 CONTROL-FIELD        PIC X.
            02 PART_ID              PIC X(8).
            02 PART_DESC            PIC X(50).
            02 PART_STATUS          PIC X(1).
            02 PART_PRICE           PIC 9(6)V9(3).
            02 PART_COST            PIC 9(6)V9(3).
            02 PART_SUPPORT         PIC X(2).
01     MAKE-CLASS-RECORD.
            02  CONTROL-FIELD       PIC X.
            02  CLASS_CODE          PIC XX.
            02  CLASS_DESC          PIC X(20).
            02  CLASS_STATUS        PIC X.
FD      BUY-FILE
        RECORD VARYING FROM 24 TO 80 CHARACTERS.
01      BUY-PART-RECORD.
            02 CONTROL-FIELD        PIC X.
            02 PART_ID              PIC X(8).
            02 PART_DESC            PIC X(50).
            02 PART_STATUS          PIC X(1).
            02 PART_PRICE           PIC 9(6)V9(3).
            02 PART_COST            PIC 9(6)V9(3).
            02 PART_SUPPORT         PIC X(2).
01      BUY-CLASS-RECORD.
            02 CONTROL-FIELD        PIC X.
            02  CLASS_CODE          PIC XX.
            02  CLASS_DESC          PIC X(20).
            02  CLASS_STATUS        PIC X.
FD      COMPONENT-FILE
        LABEL RECORDS ARE STANDARD.
01      COMPONENT-RECORD.
            02  COMP_SUB_PART      PIC X(8).
            02  COMP_OWNER_PART    PIC X(8).
            02  COMP_MEASURE       PIC X.
            02  COMP_QUANTITY      PIC 9(5).
FD      VENDOR-FILE
        LABEL RECORDS ARE STANDARD.
01      VENDOR-RECORD.
            02 VEND_ID              PIC X(8).
            02 VEND_NAME            PIC X(40).
            02 VEND_CONTACT         PIC X(30).
            02 VEND_ADD OCCURS 3 TIMES
                                   PIC X(15).
            02 VEND_PHONE           PIC 9(10).
FD      SUPPLY-FILE
        RECORD VARYING FROM 37 TO 64 CHARACTERS.
01      SUPPLY-RECORD.
            02  CONTROL-FIELD     PIC X.
            02  PART-ID           PIC X(8).
            02  VEND-NAME         PIC X(40).
```

```
            02  SUP_RATING        PIC X.
            02  SUP_TYPE          PIC X(4).
            02  SUP_LAG_TIME      PIC X(10).
01      QUOTE-RECORD.
            02  CONTROL-FIELD      PIC X.
            02  QUOTE_ID           PIC X(7).
            02  QUOTE_DATE         PIC 9(6).
            02  QUOTE_MIN_ORDER    PIC X(5).
            02  QUOTE_UNIT_PRIC    PIC 9(6)V9(3).
            02  QUOTE_QTY_PRICE    PIC 9(6)V9(3).
FD      EMPLOYEE-FILE
        LABEL RECORDS ARE STANDARD.
01      EMPLOYEE-RECORD.
            02 EMP_ID              PIC 9(5).
            02 EMP_NAME.
                03 EMP_LAST_NAME   PIC X(20).
                03 EMP_FIRST_NAME  PIC X(10).
            02 EMP_PHONE           PIC X(7).
            02 EMP_LOC             PIC X(5).
FD      DIVISION-FILE
        RECORD VARYING FROM 6 TO 26 CHARACTERS.
01      MANAGES-RECORD.
            02  CONTROL-FIELD              PIC X.
            02  GROUP_NAME                 PIC X(20).
            02  EMP_ID                     PIC 9(5).
01      CONSISTS-RECORD.
            02  CONTROL-FIELD              PIC X.
            02  EMP_ID                     PIC 9(5).
FD      RESP-FOR-FILE
        LABEL RECORDS ARE STANDARD.
01      RESP-FOR-RECORD.
            02  EMP_ID                     PIC 9(5).
            02  PART_ID                    PIC X(8).

WORKING-STORAGE SECTION.

77      ITEM-USED                PIC X(70).
77      STAT                     PIC 9(9) USAGE COMP.
77      DB-TEMP                  PIC 9(9) USAGE IS COMP.
77      CLASS-COUNT              PIC 999 VALUE IS 0.
77      PART-COUNT               PIC 999 VALUE IS 0.
77      COMPONENT-COUNT          PIC 999 VALUE IS 0.
77      VENDOR-COUNT             PIC 999 VALUE IS 0.
77      SUPPLY-COUNT             PIC 999 VALUE IS 0.
77      QUOTE-COUNT              PIC 999 VALUE IS 0.
77      EMPLOYEE-COUNT           PIC 999 VALUE IS 0.
77      DIVISION-COUNT           PIC 999 VALUE IS 0.

PROCEDURE DIVISION.

DECLARATIVES.
100-DATABASE-EXCEPTIONS SECTION.
    USE FOR DB-EXCEPTION ON OTHER.
100-PROCEDURE.
    DISPLAY "DATABASE EXCEPTION CONDITION".
    PERFORM 150-DISPLAY-MESSAGE.

150-DISPLAY-MESSAGE.
```

```
*
*  DBM$SIGNAL displays diagnostic messages based on the
*  status code in DB-CONDITION.
*
    CALL "DBM$SIGNAL".
    ROLLBACK.
    STOP RUN.
END DECLARATIVES.

DB-PROCESSING SECTION.

INITIALIZATION-ROUT.
    READY EXCLUSIVE UPDATE.

CONTROL-ROUT.
    OPEN INPUT MAKE-FILE.
    PERFORM MAKE-LOAD THRU MAKE-LOAD-END.
    CLOSE MAKE-FILE.
*    DISPLAY " ".
*    DISPLAY CLASS-COUNT, " CLASS records loaded from MAKE".
*    DISPLAY PART-COUNT, " PART records loaded from MAKE".

    OPEN INPUT BUY-FILE.
    MOVE 0 TO CLASS-COUNT.
    MOVE 0 TO PART-COUNT.
    PERFORM BUY-LOAD THRU BUY-LOAD-END.
    CLOSE BUY-FILE.
*    DISPLAY " ".
*    DISPLAY CLASS-COUNT, " CLASS records loaded from BUY".
*    DISPLAY PART-COUNT, " PART records loaded from BUY".

    OPEN INPUT VENDOR-FILE.
    PERFORM VENDOR-LOAD THRU VENDOR-LOAD-END.
    CLOSE VENDOR-FILE.
*    DISPLAY " ".
*    DISPLAY VENDOR-COUNT, " VENDOR records loaded".

    OPEN INPUT COMPONENT-FILE.
    PERFORM COMPONENT-LOAD THRU COMPONENT-LOAD-END.
    CLOSE COMPONENT-FILE.
*    DISPLAY " ".
*    DISPLAY COMPONENT-COUNT, " COMPONENT records loaded".

    OPEN INPUT EMPLOYEE-FILE.
    PERFORM EMPLOYEE-LOAD THRU EMPLOYEE-LOAD-END.
    CLOSE EMPLOYEE-FILE.
*    DISPLAY " ".
*    DISPLAY EMPLOYEE-COUNT, " EMPLOYEE records loaded".

    OPEN INPUT SUPPLY-FILE.
    PERFORM SUPPLY-LOAD THRU SUPPLY-LOAD-END.
    CLOSE SUPPLY-FILE.
*    DISPLAY " ".
*    DISPLAY SUPPLY-COUNT, " SUPPLY records loaded".
*    DISPLAY QUOTE-COUNT, " QUOTE records loaded".

    OPEN INPUT DIVISION-FILE.
    PERFORM DIVISION-LOAD THRU DIVISION-LOAD-END.
```

```
    CLOSE DIVISION-FILE.
*    DISPLAY " ".
*    DISPLAY DIVISION-COUNT, " DIVISION records loaded".

    OPEN INPUT RESP-FOR-FILE.
    PERFORM RESP-FOR-LOAD THRU RESP-FOR-LOAD-END.
    CLOSE RESP-FOR-FILE.

    COMMIT.
    STOP RUN.

MAKE-LOAD.
    READ MAKE-FILE AT END GO TO MAKE-LOAD-END.
    IF CONTROL-FIELD OF MAKE-PART-RECORD = "C"
        MOVE CORR MAKE-CLASS-RECORD TO CATEGORY
        STORE CATEGORY WITHIN MAKE
        ADD 1 TO CLASS-COUNT
            ELSE
            MOVE CORR MAKE-PART-RECORD TO PART
            STORE PART WITHIN MAKE
            ADD 1 TO PART-COUNT.
    GO TO MAKE-LOAD.

MAKE-LOAD-END.
    EXIT.

BUY-LOAD.
    READ BUY-FILE AT END GO TO BUY-LOAD-END.
    IF CONTROL-FIELD OF BUY-PART-RECORD = "C"
        MOVE CORR BUY-CLASS-RECORD TO CATEGORY
        STORE CATEGORY WITHIN BUY
        ADD 1 TO CLASS-COUNT
            ELSE
            MOVE CORR BUY-PART-RECORD TO PART
            STORE PART WITHIN BUY
            ADD 1 TO PART-COUNT.
    GO TO BUY-LOAD.

BUY-LOAD-END.
    EXIT.

VENDOR-LOAD.
    READ VENDOR-FILE AT END GO TO VENDOR-LOAD-END.
    MOVE VEND_ID OF VENDOR-RECORD TO VEND_ID OF VENDOR.
    MOVE VEND_NAME OF VENDOR-RECORD TO VEND_NAME OF VENDOR.
    MOVE VEND_CONTACT OF VENDOR-RECORD TO VEND_CONTACT OF VENDOR.
    MOVE VEND_ADD (1) TO VEND_ADDRESS (1).
    MOVE VEND_ADD (2) TO VEND_ADDRESS (2).
    MOVE VEND_ADD (3) TO VEND_ADDRESS (3).
    MOVE VEND_PHONE OF VENDOR-RECORD TO VEND_PHONE OF VENDOR.
    STORE VENDOR.
    ADD 1 TO VENDOR-COUNT.
    GO TO VENDOR-LOAD.

VENDOR-LOAD-END.
    EXIT.

COMPONENT-LOAD.
```

```
    READ COMPONENT-FILE AT END GO TO COMPONENT-LOAD-END.
    IF COMP_OWNER_PART OF COMPONENT-RECORD =
       COMP_OWNER_PART OF COMPONENT
           GO TO COMPONENT-SUB-LOAD.
    MOVE COMP_OWNER_PART OF COMPONENT-RECORD TO PART_ID OF PART.
    FIND FIRST PART WITHIN ALL_PARTS USING PART_ID OF PART
        AT END DISPLAY PART_ID OF PART,
           "COMP_OWNER_PART does not exist for COMPONENT"
           GO TO COMPONENT-LOAD.

COMPONENT-SUB-LOAD.
    MOVE COMP_SUB_PART OF COMPONENT-RECORD TO PART_ID OF PART.
    FIND FIRST PART WITHIN ALL_PARTS USING PART_ID OF PART
        RETAINING PART_USES
        AT END DISPLAY PART_ID OF PART,
            "COMP_SUB_PART does not exist for COMPONENT"
           GO TO COMPONENT-LOAD.
    MOVE CORR COMPONENT-RECORD TO COMPONENT.
    STORE COMPONENT.
    ADD 1 TO COMPONENT-COUNT.
    GO TO COMPONENT-LOAD.

COMPONENT-LOAD-END.
    EXIT.

EMPLOYEE-LOAD.
    READ EMPLOYEE-FILE AT END GO TO EMPLOYEE-LOAD-END.
    MOVE CORR EMPLOYEE-RECORD TO EMPLOYEE.
    STORE EMPLOYEE.
    ADD 1 TO EMPLOYEE-COUNT.
    GO TO EMPLOYEE-LOAD.

EMPLOYEE-LOAD-EXIT
    EXIT.

SUPPLY-LOAD.
    READ SUPPLY-FILE AT END GO TO SUPPLY-LOAD-END.

SUPPLY-LOAD-LOOP.
    IF CONTROL-FIELD OF SUPPLY-RECORD = "S"
        MOVE PART-ID OF SUPPLY-RECORD TO PART_ID OF PART
        FIND FIRST PART WITHIN ALL_PARTS USING PART_ID OF PART
          AT END
            DISPLAY PART_ID OF PART,
                    " PART-ID for SUPPLY does not exist"
            MOVE " " TO CONTROL-FIELD OF SUPPLY-RECORD
            PERFORM BAD-SUPPLY THRU BAD-SUPPLY-END
                    UNTIL CONTROL-FIELD OF SUPPLY-RECORD = "S"
            GO TO SUPPLY-LOAD-LOOP
          END-FIND
        MOVE VEND-NAME OF SUPPLY-RECORD TO VEND_NAME OF VENDOR
        FIND FIRST VENDOR WITHIN ALL_VENDORS USING VEND_NAME OF VENDOR
          AT END
            DISPLAY VEND_NAME OF VENDOR
                    "VEND-NAME for SUPPLY does not exist"
            MOVE " " TO CONTROL-FIELD OF SUPPLY-RECORD
            PERFORM BAD-SUPPLY THRU BAD-SUPPLY-END
                    UNTIL CONTROL-FIELD OF SUPPLY-RECORD = "S"
```

```
            GO TO SUPPLY-LOAD-LOOP
          END-FIND
        MOVE CORR SUPPLY-RECORD TO SUPPLY
        STORE SUPPLY
        ADD 1 TO SUPPLY-COUNT
        GO TO SUPPLY-LOAD
    ELSE
        MOVE CORR QUOTE-RECORD TO PR_QUOTE
        STORE PR_QUOTE
        ADD 1 TO QUOTE-COUNT
        GO TO SUPPLY-LOAD.

BAD-SUPPLY.
    READ SUPPLY-FILE AT END GO TO SUPPLY-LOAD-END.
    IF CONTROL-FIELD OF SUPPLY-RECORD = "Q"
        DISPLAY QUOTE_ID OF QUOTE-RECORD, " QUOTE_ID not stored".

BAD-SUPPLY-END.
    EXIT.

SUPPLY-LOAD-END.
    EXIT.

DIVISION-LOAD.
    READ DIVISION-FILE AT END GO TO DIVISION-LOAD-END.

DIVISION-LOAD-LOOP.
    IF CONTROL-FIELD OF MANAGES-RECORD = "M"
        MOVE EMP_ID OF MANAGES-RECORD TO EMP_ID OF EMPLOYEE
        FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES
                   USING EMP_ID OF EMPLOYEE
          AT END DISPLAY EMP_ID OF EMPLOYEE,
                 " EMP_ID for MANAGES does not exist"
                 MOVE " " TO CONTROL-FIELD OF MANAGES-RECORD
                 PERFORM BAD-DIVISION THRU BAD-DIVISION-END UNTIL
                 CONTROL-FIELD OF MANAGES-RECORD = "M"
                 GO TO DIVISION-LOAD-LOOP
          END-FIND
        MOVE CORR MANAGES-RECORD TO WK_GROUP
        STORE WK_GROUP
        ADD 1 TO DIVISION-COUNT
        GO TO DIVISION-LOAD
    ELSE
        MOVE EMP_ID OF CONSISTS-RECORD TO EMP_ID OF EMPLOYEE
        FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING
         EMP_ID OF EMPLOYEE
          AT END DISPLAY EMP_ID OF CONSISTS-RECORD,
            " EMP_ID for CONSISTS_OF does not exist"
            GO TO DIVISION-LOAD
          END-FIND
        CONNECT EMPLOYEE TO CONSISTS_OF
        GO TO DIVISION-LOAD.

BAD-DIVISION.
    READ DIVISION-FILE AT END GO TO DIVISION-LOAD-END.
    IF CONTROL-FIELD OF MANAGES-RECORD = "C"
        DISPLAY EMP_ID OF CONSISTS-RECORD, " EMP_ID not connected".
```

```
BAD-DIVISION-END.
    EXIT.

DIVISION-LOAD-END.
    EXIT.

RESP-FOR-LOAD.
    READ RESP-FOR-FILE AT END GO TO RESP-FOR-LOAD-END.

RESP-FOR-LOAD-LOOP.
    MOVE EMP_ID OF RESP-FOR-RECORD TO EMP_ID OF EMPLOYEE.
    FETCH FIRST EMPLOYEE WITHIN ALL_EMPLOYEES
                USING EMP_ID OF EMPLOYEE
      AT END
        DISPLAY EMP_ID OF RESP-FOR-RECORD,
        " EMP_ID for RESPONSIBLE_FOR does not exist"
        GO TO RESP-FOR-LOAD.

RESP-PART-LOOP.
    MOVE PART_ID OF RESP-FOR-RECORD TO PART_ID OF PART.
    FIND FIRST PART WITHIN ALL_PARTS USING PART_ID OF PART
      AT END
        DISPLAY PART_ID OF RESP-FOR-RECORD,
        " PART_ID for RESPONSIBLE_FOR does not exist"
        GO TO RESP-FOR-LOAD.
    CONNECT PART TO RESPONSIBLE_FOR.
    READ RESP-FOR-FILE AT END GO TO RESP-FOR-LOAD-END.
    IF EMP_ID OF RESP-FOR-RECORD = EMP_ID OF EMPLOYEE
        GO TO RESP-PART-LOOP
    ELSE
        GO TO RESP-FOR-LOAD-LOOP.
RESP-FOR-LOAD-END.
    EXIT.
```

8.2. Backing Up a Database

The PARTSBACK program in Example 8.2 unloads all PARTS database records, independently of their pointers, into a series of sequential data files. It is the first step in restructuring and reorganizing a database. For example, after backing up the database, you can change its contents. You can also create a new version of the database including different keys or new set relationships.

The PARTS database consists of a NEW root file with a default extension of .ROO describing the database instance and a series of .DBS storage files containing the actual data records. PARTS is the schema relative to the current position in CDD/Repository when the program is compiled. In the DB statement, PARTS and NEW can be logical names. If PARTS is not a logical name, VSI COBOL for OpenVMS appends PARTS to CDD$DEFAULT; for example, CDD$DEFAULT.PARTS. If NEW is not a logical name, the DBCS appends .ROO as the default file type; for example, NEW.ROO.

Example 8.2. Backing Up a Database

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     PARTSBACK.
*************************************************************
*
*  This program unloads the PARTS database
*
*************************************************************
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
        SELECT MAKE-FILE
                ASSIGN TO "DBM$PARTS:DBMMAKE.DAT".
        SELECT BUY-FILE
               ASSIGN TO "DBM$PARTS:DBMBUY.DAT".
        SELECT VENDOR-FILE
               ASSIGN TO "DBM$PARTS:DBMVENDOR.DAT".
        SELECT EMPLOYEE-FILE
               ASSIGN TO "DBM$PARTS:DBMEMPLOY.DAT".
        SELECT COMPONENT-FILE
               ASSIGN TO "DBM$PARTS:DBMCOMPON.DAT".
        SELECT SUPPLY-FILE
               ASSIGN TO "DBM$PARTS:DBMSUPPLY.DAT".
        SELECT DIVISION-FILE
               ASSIGN TO "DBM$PARTS:DBMSUPER.DAT".
        SELECT RESP-FOR-FILE
               ASSIGN TO "DBM$PARTS:DBMRESPON.DAT".

DATA DIVISION.

SUB-SCHEMA SECTION.
DB PARTSS1 WITHIN PARTS FOR NEW.

FILE SECTION.

FD      MAKE-FILE
        RECORD VARYING FROM 24 TO 80 CHARACTERS.
01      MAKE-PART-RECORD.
        02 CONTROL-FIELD        PIC X.
        02 PART_ID              PIC X(8).
        02 PART_DESC            PIC X(50).
        02 PART_STATUS          PIC X(1).
        02 PART_PRICE           PIC 9(6)V9(3).
        02 PART_COST            PIC 9(6)V9(3).
        02 PART_SUPPORT         PIC X(2).
01      MAKE-CLASS-RECORD.
        02  CONTROL-FIELD       PIC X.
        02  CLASS_CODE          PIC XX.
        02  CLASS_DESC          PIC X(20).
        02  CLASS_STATUS        PIC X.

FD      BUY-FILE
        RECORD VARYING FROM 24 TO 80 CHARACTERS.
01      BUY-PART-RECORD.
        02 CONTROL-FIELD        PIC X.
        02 PART_ID              PIC X(8).
        02 PART_DESC            PIC X(50).
        02 PART_STATUS          PIC X(1).
        02 PART_PRICE           PIC 9(6)V9(3).
        02 PART_COST            PIC 9(6)V9(3).
        02 PART_SUPPORT         PIC X(2).
01      BUY-CLASS-RECORD.
```

```
        02 CONTROL-FIELD        PIC X.
        02 CLASS_CODE           PIC XX.
        02 CLASS_DESC           PIC X(20).
        02 CLASS_STATUS         PIC X.

FD      COMPONENT-FILE
        LABEL RECORDS ARE STANDARD.
01      COMPONENT-RECORD.
        02  COMP_SUB_PART      PIC X(8).
        02  COMP_OWNER_PART    PIC X(8).
        02  COMP_MEASURE       PIC X.
        02  COMP_QUANTITY      PIC 9(5).

FD      VENDOR-FILE
        LABEL RECORDS ARE STANDARD.
01      VENDOR-RECORD.
        02 VEND_ID                     PIC X(8).
        02 VEND_NAME                   PIC X(40).
        02 VEND_CONTACT                PIC X(30).
        02 VEND_ADDRESS OCCURS 3 TIMES PIC X(15).
        02 VEND_PHONE                  PIC 9(10).

FD      SUPPLY-FILE
        RECORD VARYING FROM 37 TO 64 CHARACTERS.
01      SUPPLY-RECORD.
        02  CONTROL-FIELD      PIC X.
        02  PART-ID            PIC X(8).
        02  VEND-NAME          PIC X(40).
        02  SUP_RATING         PIC X.
        02  SUP_TYPE           PIC X(4).
        02  SUP_LAG_TIME       PIC X(10).
01      QUOTE-RECORD.
        02  CONTROL-FIELD      PIC X.
        02  QUOTE_ID           PIC X(7).
        02  QUOTE_DATE         PIC 9(6).
        02  QUOTE_MIN_ORDER    PIC X(5).
        02  QUOTE_UNIT_PRIC    PIC 9(6)V9(3).
        02  QUOTE_QTY_PRICE    PIC 9(6)V9(3).

FD      EMPLOYEE-FILE
        LABEL RECORDS ARE STANDARD.
01      EMPLOYEE-RECORD.
        02 EMP_ID                      PIC 9(5).
        02 EMP_NAME.
                03 EMP_LAST_NAME       PIC X(20).
                03 EMP_FIRST_NAME      PIC X(10).
        02 EMP_PHONE                   PIC X(7).
        02 EMP_LOC                     PIC X(5).

FD      DIVISION-FILE
        RECORD VARYING FROM 6 TO 26 CHARACTERS.
01      MANAGES-RECORD.
        02  CONTROL-FIELD       PIC X.
        02  GROUP_NAME          PIC X(20).
        02  EMP_ID              PIC 9(5).
01      CONSISTS-RECORD.
        02  CONTROL-FIELD       PIC X.
        02  EMP_ID              PIC 9(5).
```

```
FD      RESP-FOR-FILE
        LABEL RECORDS ARE STANDARD.
01      RESP-FOR-RECORD.
        02  EMP_ID             PIC 9(5).
        02  PART_ID            PIC X(8).

WORKING-STORAGE SECTION.

77      CLASS-COUNT             PIC 999 VALUE IS 0.
77      PART-COUNT              PIC 999 VALUE IS 0.
77      COMPONENT-COUNT         PIC 999 VALUE IS 0.
77      VENDOR-COUNT            PIC 999 VALUE IS 0.
77      SUPPLY-COUNT            PIC 999 VALUE IS 0.
77      QUOTE-COUNT             PIC 999 VALUE IS 0.
77      EMPLOYEE-COUNT          PIC 999 VALUE IS 0.

PROCEDURE DIVISION.

DECLARATIVES.
100-DATABASE-EXCEPTIONS SECTION.
    USE FOR DB-EXCEPTION ON OTHER.
100-PROCEDURE.
    DISPLAY "DATABASE EXCEPTION CONDITION".
    PERFORM 150-DISPLAY-MESSAGE.

150-DISPLAY-MESSAGE.
*
* DBM$SIGNAL displays diagnostic messages based on the
* status code in DB-CONDITION.
*
    CALL "DBM$SIGNAL".
    ROLLBACK.
    STOP RUN.
END DECLARATIVES.

DB-PROCESSING SECTION.

INITIALIZATION-ROUT.
    READY PROTECTED.

CONTROL-ROUT.
    OPEN OUTPUT COMPONENT-FILE, SUPPLY-FILE.
    OPEN OUTPUT MAKE-FILE.
    PERFORM MAKE-UNLOAD THRU MAKE-UNLOAD-END.
    CLOSE MAKE-FILE.
    DISPLAY " ".
    DISPLAY CLASS-COUNT, " CLASS records unloaded from MAKE".
    DISPLAY PART-COUNT, " PART records unloaded from MAKE".

    OPEN OUTPUT BUY-FILE.
    MOVE 0 TO CLASS-COUNT.
    MOVE 0 TO PART-COUNT.
    PERFORM BUY-UNLOAD THRU BUY-UNLOAD-END.
    CLOSE BUY-FILE, COMPONENT-FILE, SUPPLY-FILE.
    DISPLAY " ".
    DISPLAY CLASS-COUNT, " CLASS records unloaded from BUY".
    DISPLAY PART-COUNT, " PART records unloaded from BUY".
```

```
    DISPLAY " ".
    DISPLAY SUPPLY-COUNT, " SUPPLY records unloaded".
    DISPLAY QUOTE-COUNT, " QUOTE records unloaded".
    DISPLAY COMPONENT-COUNT " COMPONENT records unloaded".

    OPEN OUTPUT VENDOR-FILE.
    PERFORM VENDOR-UNLOAD THRU VENDOR-UNLOAD-END.
    CLOSE VENDOR-FILE.
    DISPLAY " ".
    DISPLAY VENDOR-COUNT, " VENDOR records unloaded".

    OPEN OUTPUT EMPLOYEE-FILE, RESP-FOR-FILE, DIVISION-FILE.
    PERFORM EMPLOYEE-UNLOAD THRU EMPLOYEE-UNLOAD-END.
    CLOSE EMPLOYEE-FILE, RESP-FOR-FILE, DIVISION-FILE.
    DISPLAY " ".
    DISPLAY EMPLOYEE-COUNT, " EMPLOYEE records unloaded".

    COMMIT.
    STOP RUN.

MAKE-UNLOAD.
    FETCH NEXT CATEGORY WITHIN MAKE
        AT END GO TO MAKE-UNLOAD-END.
    MOVE "C" TO CONTROL-FIELD OF MAKE-CLASS-RECORD.
    MOVE CORR CATEGORY TO MAKE-CLASS-RECORD.
    ADD 1 TO CLASS-COUNT.
    WRITE MAKE-CLASS-RECORD.

MAKE-PART-LOOP.
    FETCH NEXT PART WITHIN CLASS_PART RETAINING REALM
        AT END GO TO MAKE-UNLOAD.
    MOVE "P" TO CONTROL-FIELD OF MAKE-PART-RECORD.
    MOVE CORR PART TO MAKE-PART-RECORD.
    ADD 1 TO PART-COUNT.
    WRITE MAKE-PART-RECORD.
    PERFORM COMPONENT-SUPPLY-UNLOAD THRU
            COMPONENT-SUPPLY-UNLOAD-END.
    GO TO MAKE-PART-LOOP.

MAKE-UNLOAD-END.
    EXIT.

BUY-UNLOAD.
    FETCH NEXT CATEGORY WITHIN BUY
        AT END GO TO BUY-UNLOAD-END.
    MOVE "C" TO CONTROL-FIELD OF BUY-CLASS-RECORD.
    MOVE CORR CATEGORY TO BUY-CLASS-RECORD.
    ADD 1 TO CLASS-COUNT.
    WRITE BUY-CLASS-RECORD.

BUY-PART-LOOP.
    FETCH NEXT PART WITHIN CLASS_PART RETAINING REALM
        AT END GO TO BUY-UNLOAD.
    MOVE "P" TO CONTROL-FIELD OF BUY-PART-RECORD.
    MOVE CORR PART TO BUY-PART-RECORD.
    ADD 1 TO PART-COUNT.
    WRITE BUY-PART-RECORD.
    PERFORM COMPONENT-SUPPLY-UNLOAD THRU
```

```
            COMPONENT-SUPPLY-UNLOAD-END.
    GO TO BUY-PART-LOOP.

BUY-UNLOAD-END.
    EXIT.

COMPONENT-SUPPLY-UNLOAD.

COMPONENT-UNLOAD.
    FETCH NEXT COMPONENT WITHIN PART_USES RETAINING REALM
        AT END GO TO SUPPLY-QUOTE-LOOP.
    MOVE CORR COMPONENT TO COMPONENT-RECORD.
    ADD 1 TO COMPONENT-COUNT.
    WRITE COMPONENT-RECORD.
    GO TO COMPONENT-UNLOAD.

SUPPLY-QUOTE-LOOP.
    FETCH NEXT WITHIN PART_INFO RETAINING REALM
        AT END GO TO COMPONENT-SUPPLY-UNLOAD-END.
    IF DB-CURRENT-RECORD-NAME = "PR_QUOTE" THEN
        MOVE CORR PR_QUOTE TO QUOTE-RECORD
        MOVE "Q" TO CONTROL-FIELD OF QUOTE-RECORD
        ADD 1 TO QUOTE-COUNT
        WRITE QUOTE-RECORD
        GO TO SUPPLY-QUOTE-LOOP
    ELSE
        MOVE CORR SUPPLY TO SUPPLY-RECORD
        FETCH OWNER WITHIN VENDOR_SUPPLY
        MOVE "S" TO CONTROL-FIELD OF SUPPLY-RECORD
        MOVE VEND_NAME OF VENDOR TO VEND-NAME OF SUPPLY-RECORD
        MOVE PART_ID OF PART TO PART-ID OF SUPPLY-RECORD
        ADD 1 TO SUPPLY-COUNT
        WRITE SUPPLY-RECORD
        GO TO SUPPLY-QUOTE-LOOP.

COMPONENT-SUPPLY-UNLOAD-END.
    EXIT.

VENDOR-UNLOAD.
    FREE CURRENT WITHIN MARKET.

VENDOR-UNLOAD-LOOP.
    FETCH NEXT VENDOR WITHIN MARKET
        AT END GO TO VENDOR-UNLOAD-END.
    ADD 1 TO VENDOR-COUNT.
    MOVE VEND_ID OF VENDOR TO VEND_ID OF VENDOR-RECORD.
    MOVE VEND_NAME OF VENDOR TO VEND_NAME OF VENDOR-RECORD.
    MOVE VEND_CONTACT OF VENDOR TO VEND_CONTACT OF VENDOR-RECORD.
    MOVE VEND_ADDRESS OF VENDOR (1) TO
         VEND_ADDRESS OF VENDOR-RECORD (1).
    MOVE VEND_ADDRESS OF VENDOR (2) TO
         VEND_ADDRESS OF VENDOR-RECORD (2).
    MOVE VEND_ADDRESS OF VENDOR (3) TO
         VEND_ADDRESS OF VENDOR-RECORD (3).
    MOVE VEND_PHONE OF VENDOR TO VEND_PHONE OF VENDOR-RECORD.
    WRITE VENDOR-RECORD.
    GO TO VENDOR-UNLOAD-LOOP.
```

```
VENDOR-UNLOAD-END.
    EXIT.

EMPLOYEE-UNLOAD.
    FETCH NEXT EMPLOYEE WITHIN ALL_EMPLOYEES
        AT END GO TO EMPLOYEE-UNLOAD-END.
    MOVE CORR EMPLOYEE TO EMPLOYEE-RECORD.
    ADD 1 TO EMPLOYEE-COUNT.
    WRITE EMPLOYEE-RECORD.

DIVISION-UNLOAD.
    FETCH NEXT WITHIN MANAGES
        AT END GO TO RESP-UNLOAD.
    MOVE EMP_ID OF EMPLOYEE TO EMP_ID OF MANAGES-RECORD.
    MOVE GROUP_NAME OF WK_GROUP TO GROUP_NAME OF MANAGES-RECORD.
    MOVE "M" TO CONTROL-FIELD OF MANAGES-RECORD.
    WRITE MANAGES-RECORD.

CONSISTS-UNLOAD.
    FETCH NEXT WITHIN CONSISTS_OF RETAINING MANAGES ALL_EMPLOYEES
        AT END GO TO DIVISION-UNLOAD.
    MOVE "C" TO CONTROL-FIELD OF CONSISTS-RECORD.
    MOVE EMP_ID OF EMPLOYEE TO  EMP_ID OF CONSISTS-RECORD.
    WRITE CONSISTS-RECORD.
    GO TO CONSISTS-UNLOAD.

RESP-UNLOAD.
    FETCH CURRENT WITHIN ALL_EMPLOYEES.
RESP-UNLOAD-LOOP.
    FETCH NEXT WITHIN RESPONSIBLE_FOR
        AT END GO TO EMPLOYEE-UNLOAD.
    MOVE PART_ID OF PART TO PART_ID OF RESP-FOR-RECORD.
    MOVE EMP_ID OF EMPLOYEE TO EMP_ID OF RESP-FOR-RECORD.
    WRITE RESP-FOR-RECORD.
    GO TO RESP-UNLOAD-LOOP.
EMPLOYEE-UNLOAD-END.
    EXIT.
```

8.3. Accessing and Displaying Database Information

The PARTBOM program in Example 8.3 produces a report of subcomponents (bill of materials) for a part in the PARTS database. Refer to Example 5.3 for an explanation of the report and Section 8.6 for a sample listing.

Example 8.3. Accessing and Displaying Database Information

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PARTBOM.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.


DATA DIVISION.
SUB-SCHEMA SECTION.
```

```
DB    PARTSS1 WITHIN PARTS FOR NEW.
LD    KEEP-COMPONENT.

WORKING-STORAGE SECTION.

01      INPUT-REC               PIC X(80).

01      INDENT-LEVEL            PIC 9(02)  VALUE 40.
01      END-OF-COLLECTION       PIC 9(01)  VALUE 0.
        88  END-COLLECTION                VALUE 1.

01      INDENT-TREE.
        02  INDENT-TREE-ARRAY   PIC X(03)  OCCURS 1 TO 40 TIMES
                                DEPENDING ON INDENT-LEVEL.
PROCEDURE DIVISION.

INITIALIZATION.
    READY   MAKE, BUY EXCLUSIVE RETRIEVAL.
    MOVE    ALL "|  " TO INDENT-TREE.

SOLICIT-INPUT.
    MOVE ZERO TO END-OF-COLLECTION.
    DISPLAY    " ".
    DISPLAY    "Enter PART_ID> " WITH NO ADVANCING.
    MOVE    SPACES TO INPUT-REC.
    ACCEPT PART_ID
        AT END GO TO PARTBOM-DONE.
    FETCH    FIRST PART WITHIN ALL_PARTS USING PART_ID
        AT END DISPLAY "***  Part number ",

                                PART_ID, " not found.  ***"
               GO TO SOLICIT-INPUT.
    DISPLAY    " ".
    DISPLAY    " ".
    DISPLAY "+-----------------------------------+".
    DISPLAY "| Parts Bill of Materials Explosion |".
    DISPLAY "|          (COBOL Version)          |".
    DISPLAY "|         Part-id: " PART_ID "      |".
    DISPLAY "+-----------------------------------+".
    DISPLAY " ".
    DISPLAY " ".
    DISPLAY " ".
    DISPLAY    PART_ID, " - ", PART_DESC
    MOVE ZERO TO INDENT-LEVEL.
    FREE ALL FROM KEEP-COMPONENT.
    PERFORM PARTBOM-LOOP THRU PARTBOM-LOOP-EXIT
        UNTIL END-COLLECTION.
    GO TO SOLICIT-INPUT.

PARTBOM-DONE.
    COMMIT.
    DISPLAY " ".
    DISPLAY "END COBOL PARTBOM.".
    STOP RUN.

PARTBOM-LOOP.
    FIND NEXT COMPONENT WITHIN PART_USES
```

```
        AT END PERFORM POP-COMPONENT THRU POP-COMPONENT-EXIT
               GO TO PARTBOM-LOOP-EXIT.
    KEEP CURRENT USING KEEP-COMPONENT.
    ADD 1 TO INDENT-LEVEL.
    FIND OWNER PART_USED_ON.
    GET PART_ID, PART_DESC.
    DISPLAY INDENT-TREE, PART_ID, " - ", PART_DESC.

PARTBOM-LOOP-EXIT.
    EXIT.

POP-COMPONENT.
    FIND LAST WITHIN KEEP-COMPONENT
        AT END MOVE 1 TO END-OF-COLLECTION
               GO TO POP-COMPONENT-EXIT.
    FREE LAST WITHIN KEEP-COMPONENT.
    SUBTRACT 1 FROM INDENT-LEVEL.

POP-COMPONENT-EXIT.
    EXIT.
```

8.4. PARTBOM Sample Run

Example 8.4 displays a sample run of the PARTBOM program in Example 8.3.

Example 8.4. Sample Run of the PARTBOM Program

```
Enter PARTID> BT163456

                 +-----------------------------------+
                 | Parts Bill of Materials Explosion |
                 |          (COBOL Version)          |
                 |         Part-id: BT163456         |
                 +-----------------------------------+

BT163456 - VT100
|  BU355678 - VT100 NON REFLECTIVE SCREEN
|  BU345670 - TERMINAL TABLE VT100
|  |  AZ345678 - 3/4 INCH SCREWS
|  |  AZ167890 - 1/2 INCH SCREWS
|  |  AZ517890 - 1/4 INCH BOLTS
|  |  AZ012345 - 3 INCH NAILS
|  |  AS234567 - 1/4 INCH TACKS
|  |  AS901234 - 3/8 INCH SCREWS
|  |  AS456789 - 4/5 INCH CLAMP
|  |  AS560890 - 1 INCH CLAMP
|  BU456789 - PLASTIC KEY ALPHA.
|  BU345438 - PLASTIC KEY NUM.
|  BU234567 - VIDEO TUBE
|  |  AZ345678 - 3/4 INCH SCREWS
|  |  AZ789012 - 3/8 INCH BOLTS
|  |  AS234567 - 1/4 INCH TACKS
|  |  AS560890 - 1 INCH CLAMP
|  BU890123 - VT100 HOUSING
|  BU876778 - VT100 SCREEN
|  AZ345678 - 3/4 INCH SCREWS
|  AZ567890 - 1/4 INCH SCREWS
```

```
|  AZ789012 - 3/8 INCH BOLTS
|  AS901234 - 3/8 INCH SCREWS
|  AS890123 - 3/4 INCH ELECTRICAL TAPE

Enter PARTID> [ctrl/z]

END COBOL PARTBOM.
```

8.5. Creating Relationships Between Records of the Same Type

The STOOL program in Example 8.5 illustrates how to create a relationship between records of the same type. It loads and connects the parts example discussed in Section 5.9.2.2 and produces a parts breakdown report illustrating the relationships. Section 8.6 contains the sample report.

Example 8.5. Creating Relationships Between Records of the Same Type

```
IDENTIFICATION DIVISION.
PROGRAM-ID. STOOL.
DATA DIVISION.
SUB-SCHEMA SECTION.
DB  PARTSS1 WITHIN  PARTS FOR "NEW.ROO".
LD  KEEP-COMPONENT.
WORKING-STORAGE SECTION.
01  DB-ERROR-CHECK       PIC 9.
    88  DB-ERROR         VALUE 1.
    88  DB-OK            VALUE 0.
01  DB-COND              PIC 9(9).
01  DB-ID                PIC 9(4).

PROCEDURE DIVISION.
A000-BEGIN.
    READY USAGE-MODE IS CONCURRENT UPDATE.
    MOVE 0 TO DB-ERROR-CHECK.
    PERFORM B000-STORE-PARTS THROUGH
            B300-BUILD-AND-STORE-STOOL-LEG.
    IF DB-OK PERFORM C000-STORE-COMPONENTS
                     THRU 800-VERIFY-ROUTINE.

A100-EOJ.
*   IF DB-ERROR
    ROLLBACK ON ERROR DISPLAY "Error on ROLLBACK"
             PERFORM 900-DISPLAY-DB-CONDITION
             END-ROLLBACK
    DISPLAY "End of Job".
    STOP RUN.

B000-STORE-PARTS.
    FIND FIRST PART ON ERROR
         DISPLAY "Positioning to first part is unsuccessful"
         PERFORM 900-DISPLAY-DB-CONDITION
         MOVE 1 TO DB-ERROR-CHECK.

B100-BUILD-AND-STORE-STOOL.
    MOVE "SAMP1" TO PART_ID.
    MOVE "STOOL" TO PART_DESC.
```

```
    MOVE "G"     TO PART_STATUS.
    MOVE 11      TO PART_PRICE.
    MOVE 6       TO PART_COST.
    MOVE SPACES  TO PART_SUPPORT.
    IF DB-OK STORE PART ON ERROR
          DISPLAY "B100 Error in storing STOOL"
          PERFORM 900-DISPLAY-DB-CONDITION
          MOVE 1 TO DB-ERROR-CHECK.

B200-BUILD-AND-STORE-STOOL-SEAT.
    MOVE "SAMP2"      TO PART_ID.
    MOVE "STOOL SEAT" TO PART_DESC.
    MOVE "G"          TO PART_STATUS.
    MOVE 3            TO PART_PRICE.
    MOVE 2            TO PART_COST.
    MOVE SPACES       TO PART_SUPPORT.
    IF DB-OK STORE PART ON ERROR
          DISPLAY "B200 Error in storing STOOL SEAT"
          PERFORM 900-DISPLAY-DB-CONDITION
          MOVE 1 TO DB-ERROR-CHECK.

B300-BUILD-AND-STORE-STOOL-LEG.
    MOVE "SAMP3"      TO PART_ID.
    MOVE "STOOL LEGS" TO PART_DESC.
    MOVE "G"          TO PART_STATUS.
    MOVE 2            TO PART_PRICE.
    MOVE 1            TO PART_COST.
    MOVE SPACES       TO PART_SUPPORT.
    IF DB-OK STORE PART ON ERROR
          DISPLAY "B300 Error in storing STOOL LEGS"
          PERFORM 900-DISPLAY-DB-CONDITION
          MOVE 1 TO DB-ERROR-CHECK.

C000-STORE-COMPONENTS.
    MOVE "STOOL" TO PART_DESC.

C100-FIND-STOOL.
    FIND FIRST PART USING PART_DESC ON ERROR
         DISPLAY "C000 Error in finding STOOL"
         PERFORM 900-DISPLAY-DB-CONDITION
         MOVE 1 TO DB-ERROR-CHECK.
    MOVE "STOOL SEAT" TO PART_DESC.

C200-FIND-STOOL-SEAT.
    IF DB-OK
       FIND FIRST PART USING PART_DESC RETAINING PART_USES
         ON ERROR
             DISPLAY "C000 Error in finding STOOL SEAT"
             PERFORM 900-DISPLAY-DB-CONDITION
             MOVE 1 TO DB-ERROR-CHECK.

C300-CONNECT-COMPONENT-1.
    MOVE "SAMP2" TO COMP_SUB_PART.
    MOVE "SAMP1" TO COMP_OWNER_PART.
    MOVE "U"     TO COMP_MEASURE.
    MOVE 1       TO COMP_QUANTITY.
    IF DB-OK
       STORE COMPONENT RETAINING PART_USES
```

```
         ON ERROR
             DISPLAY "C000 Error in storing first component"
             PERFORM 900-DISPLAY-DB-CONDITION
             MOVE 1 TO DB-ERROR-CHECK.

C400-FIND-STOOL-LEGS.
    MOVE "STOOL LEGS" TO PART_DESC.
    IF DB-OK
       FIND FIRST PART USING PART_DESC RETAINING PART_USES
         ON ERROR
             DISPLAY "C000 Error in finding STOOL LEGS"
             PERFORM 900-DISPLAY-DB-CONDITION
             MOVE 1 TO DB-ERROR-CHECK.

C500-CONNECT-COMPONENT-4.
    MOVE "SAMP3" TO COMP_SUB_PART.
    MOVE "SAMP1" TO COMP_OWNER_PART.
    MOVE "U"     TO COMP_MEASURE.
    MOVE 4       TO COMP_QUANTITY.
    IF DB-OK
       STORE COMPONENT
         ON ERROR
             DISPLAY "C000 Error in storing second component"
             PERFORM 900-DISPLAY-DB-CONDITION
             MOVE 1 TO DB-ERROR-CHECK.

800-VERIFY-ROUTINE.
    CALL "PARTBOM".

900-DISPLAY-DB-CONDITION.
    MOVE DB-CONDITION                  TO DB-COND.
    MOVE DB-CURRENT-RECORD-ID          TO DB-ID.
    DISPLAY "DB-CONDITION            - ", DB-COND.
    DISPLAY "DB-CURRENT-RECORD-NAME  - ",
                            DB-CURRENT-RECORD-NAME.
    DISPLAY "DB-CURRENT-RECORD-ID    - ", DB-ID.
    CALL "DBM$SIGNAL".

IDENTIFICATION DIVISION.
PROGRAM-ID.  PARTBOM.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT  INPUT-FILE ASSIGN TO "SYS$COMMAND".

DATA DIVISION.
SUB-SCHEMA SECTION.
*  DB PARTSS1 WITHIN  PARTS FOR "NEW.ROO".

FILE SECTION.
FD      INPUT-FILE
        LABEL RECORDS ARE STANDARD
        DATA  RECORD  IS  INPUT-REC.
01      INPUT-REC               PIC X(80).

WORKING-STORAGE SECTION.
01      INDENT-LEVEL            PIC 9(02)  VALUE 40.
01      DBM$_END                PIC 9(09)  COMP
```

```
                                VALUE EXTERNAL DBM$_END.
01      END-OF-COLLECTION       PIC 9(01)  VALUE 0.
        88  END-COLLECTION                VALUE 1.
01      INDENT-TREE.
        02  INDENT-TREE-ARRAY   PIC X(03)
                                OCCURS 1 TO 40 TIMES
                                DEPENDING  ON INDENT-LEVEL.

PROCEDURE DIVISION.

INITIALIZATION.
    OPEN INPUT  INPUT-FILE.
    MOVE ALL "|  " TO INDENT-TREE.

SOLICIT-INPUT.
    MOVE ZERO TO END-OF-COLLECTION.
    DISPLAY " ".
    DISPLAY "Enter PART_ID> " WITH NO ADVANCING.
    MOVE SPACES TO INPUT-REC.
    READ INPUT-FILE INTO PART_ID
        AT END GO TO PARTBOM-DONE.
    FETCH FIRST PART WITHIN ALL_PARTS USING PART_ID
        AT END DISPLAY "*** Part number ",
                                PART_ID, " not found.  ***"
              GO TO SOLICIT-INPUT.
    DISPLAY    " ".
    DISPLAY    " ".
    DISPLAY
    DISPLAY "+-----------------------------------+".
    DISPLAY "| Parts Bill of Materials Explosion |".
    DISPLAY "|          (COBOL Version)          |".
    DISPLAY "|         Part-id: " PART_ID "      |".
    DISPLAY "+-----------------------------------+".
    DISPLAY " ".
    DISPLAY " ".
    DISPLAY " ".
    DISPLAY    PART_ID, " - ", PART_DESC
    MOVE ZERO TO INDENT-LEVEL.
    FREE ALL FROM KEEP-COMPONENT.
    PERFORM PARTBOM-LOOP THRU PARTBOM-LOOP-EXIT
        UNTIL END-COLLECTION.
    GO TO SOLICIT-INPUT.

PARTBOM-DONE.
    CLOSE INPUT-FILE.
    EXIT PROGRAM.

PARTBOM-LOOP.
    FIND NEXT COMPONENT WITHIN PART_USES
         AT END PERFORM POP-COMPONENT
                        THRU POP-COMPONENT-EXIT
         GO TO PARTBOM-LOOP-EXIT.
    KEEP CURRENT USING KEEP-COMPONENT.
    ADD 1 TO INDENT-LEVEL.
    FIND OWNER PART_USED_ON.
    GET PART_ID, PART_DESC.
    DISPLAY INDENT-TREE, PART_ID, " - ", PART_DESC.
```

```
PARTBOM-LOOP-EXIT.
    EXIT.

POP-COMPONENT.
    FIND    LAST WITHIN KEEP-COMPONENT
        AT END MOVE 1 TO END-OF-COLLECTION
               GO TO POP-COMPONENT-EXIT.
    FREE    LAST WITHIN KEEP-COMPONENT.
    SUBTRACT 1 FROM INDENT-LEVEL.

POP-COMPONENT-EXIT.
    EXIT.
END PROGRAM PARTBOM.
END PROGRAM STOOL.
```

8.6. STOOL Program Parts Breakdown Report —Sample Run

This is the report output by the STOOL program in Example 8.5.

```
      Enter PARTID>  (SAMP1 [RET]
          +-----------------------------------+
          | Parts Bill of Materials Explosion |
          |         (COBOL Version)           |
          |          Part-id: SAMP1           |
          +-----------------------------------+

          SAMP1    - STOOL
          SAMP3    - STOOL LEGS
          SAMP2    - STOOL SEAT

          Enter PARTID> [ctrl/z]
          End of Job
```

8.7. Creating New Record Relationships

The PERSONNEL-UPDATE program in Example 8.6 creates the records and implements the relationships described in Section 5.9.2.3. It directly contains two other programs: PROMOTION-UPDATE and PERSONNEL-REPORT. PROMOTION-UPDATE is directly contained by PERSONNEL-UPDATE. It changes the record relationships created by PERSONNEL-UPDATE. PERSONNEL-REPORT is also directly contained by PERSONNEL-UPDATE. It generates one report showing the record relationships just after creation by PERSONNEL-UPDATE and another report showing the new record relationships. PERSONNEL-REPORT is a Report Writer program. Section 8.7.1 and Section 8.7.2 each contain a report generated by the PERSONNEL-UPDATE program.

Example 8.6. Creating New Record Relationships

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PERSONNEL-UPDATE.

DATA DIVISION.
SUB-SCHEMA SECTION.
DB PARTSS1 WITHIN PARTS FOR "NEW.ROO".
LD KEEPSUPER.
LD KEEP-EMPLOYEE.
```

```
WORKING-STORAGE SECTION.
01  ANSWER      PIC X.

PROCEDURE DIVISION.
A000-BEGIN.
    READY USAGE-MODE IS UPDATE.
    PERFORM A100-EMPLOYEE-LOAD.
    PERFORM A200-CONNECTING-TO-CONSISTS-OF.
    DISPLAY "Employees and groups are loaded".
    DISPLAY "Personnel Report before update ..."
    CALL "PERSONNEL-REPORT".
    DISPLAY "Press your carriage return key to continue".
    ACCEPT ANSWER.
    CALL "PROMOTION-UPDATE".
    DISPLAY "Promotions completed".
    DISPLAY "Press your carriage return key to continue".
    ACCEPT ANSWER.
    DISPLAY "Personnel Report after update ...".
    CALL "PERSONNEL-REPORT".

A010-EOJ.
    ROLLBACK.
    DISPLAY "End of PERSONNEL-UPDATE".
    STOP RUN.

A100-EMPLOYEE-LOAD.
    MOVE 10500     TO EMP_ID.
    MOVE "HOWELL"  TO EMP_LAST_NAME.
    MOVE "JOHN"    TO EMP_FIRST_NAME.
    MOVE 1111111   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 08400     TO EMP_ID.
    MOVE "NOYCE"   TO EMP_LAST_NAME.
    MOVE "BILL"    TO EMP_FIRST_NAME.
    MOVE 2222222   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 06600     TO EMP_ID.
    MOVE "MOORE"   TO EMP_LAST_NAME.
    MOVE "BRUCE"   TO EMP_FIRST_NAME.
    MOVE 3333333   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 01000     TO EMP_ID.
    MOVE "RAVAN"   TO EMP_LAST_NAME.
    MOVE "JERRY"   TO EMP_FIRST_NAME.
    MOVE 5555555   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 04000     TO EMP_ID.
    MOVE "BURLEW"  TO EMP_LAST_NAME.
    MOVE "THOMAS"  TO EMP_FIRST_NAME.
```

```
    MOVE 6666666   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 07000     TO EMP_ID.
    MOVE "NEILS"   TO EMP_LAST_NAME.
    MOVE "ALBERT"  TO EMP_FIRST_NAME.
    MOVE 7777777   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 05000     TO EMP_ID.
    MOVE "KLEIN"   TO EMP_LAST_NAME.
    MOVE "DON"     TO EMP_FIRST_NAME.
    MOVE 8888888   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 02000     TO EMP_ID.
    MOVE "DEANE"   TO EMP_LAST_NAME.
    MOVE "FRANK"   TO EMP_FIRST_NAME.
    MOVE 9999999   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 01400     TO EMP_ID.
    MOVE "RILEY"   TO EMP_LAST_NAME.
    MOVE "GEORGE"  TO EMP_FIRST_NAME.
    MOVE 1234567   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 05500     TO EMP_ID.
    MOVE "BAKER"   TO EMP_LAST_NAME.
    MOVE "DOUGH"   TO EMP_FIRST_NAME.
    MOVE 7654321   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

    MOVE 07400     TO EMP_ID.
    MOVE "FIFER"   TO EMP_LAST-NAME.
    MOVE "MIKE"    TO EMP_FIRST_NAME.
    MOVE 1212121   TO EMP_PHONE.
    MOVE "N.H."    TO EMP_LOC.
    STORE EMPLOYEE.

A200-CONNECTING-TO-CONSISTS-OF.
    MOVE 10500 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    MOVE "A" TO GROUP_NAME.
    STORE WK_GROUP.

    MOVE 08400 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 06600 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
```

```
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 08400 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    MOVE "B1" TO GROUP_NAME.
    STORE WK_GROUP.

    MOVE 01000 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 04000 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 07000 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 06600 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    MOVE "B2" TO GROUP_NAME.
    STORE WK_GROUP.

    MOVE 01400 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 02000 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 05000 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 05500 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

    MOVE 07400 TO EMP_ID.
    FIND FIRST EMPLOYEE WITHIN ALL_EMPLOYEES USING EMP_ID.
    CONNECT EMPLOYEE TO CONSISTS_OF.

IDENTIFICATION DIVISION.
PROGRAM-ID. PROMOTION-UPDATE.

PROCEDURE DIVISION.
A000-BEGIN.
    MOVE "A" TO GROUP_NAME.
*
* The next statement makes HOWELL's GROUP "A" record current
*
    FIND FIRST WK_GROUP USING GROUP_NAME.
*
* The next two statements fetch KLEIN using EMP_ID.
* The RETAINING clause keeps the WK_GROUP record "A"
* as current of the CONSISTS_OF set. This allows the program
```

```
* to connect KLEIN to the correct occurrence of WK_GROUP.
* A fetch to KLEIN without the RETAINING clause makes KLEIN
* current of CONSISTS_OF thus destroying the pointer to the
* WK_GROUP record "A".
*
    MOVE 05000 TO EMP_ID.
    FETCH FIRST EMPLOYEE USING EMP_ID RETAINING CONSISTS_OF.
*
* The next statement disconnects KLEIN from the WK_GROUP "B1"
* record and connects him to the current WK_GROUP "A" record.
*
    RECONNECT EMPLOYEE WITHIN CONSISTS_OF.
*
* The next two sentences create and store a WK_GROUP record.
* Because KLEIN is current of EMPLOYEE, a STORE WK_GROUP
* automatically connects WK_GROUP as a member of the MANAGES
* set owned by KLEIN, and makes "B3" current of the MANAGES
* and CONSISTS_OF sets.
*
    MOVE "B3" TO WK_GROUP.
    STORE WK_GROUP.
*
* The next two statements fetch NEILS and retain WK_GROUP
* "B3" as current of CONSISTS_OF.
*
    MOVE 7000 TO EMP_ID.
    FETCH FIRST EMPLOYEE USING EMP_ID RETAINING CONSISTS_OF.
*
* The next statement disconnects NEILS from WK_GROUP "B1"
* record and reconnects him to the WK_GROUP "B3" record.
* It also retains "B3" as current of CONSISTS_OF. This
* maintains the pointer at "B3" allowing the program to
* reassign RILEY to KLEIN.
*
    RECONNECT EMPLOYEE WITHIN CONSISTS_OF RETAINING CONSISTS_OF.
*
* The next three statements fetch RILEY, disconnect him from
* "B2" and reconnect him to "B3".
*
    MOVE 01400 TO EMP_ID.
    FETCH FIRST EMPLOYEE USING EMP_ID RETAINING CONSISTS_OF.
    RECONNECT EMPLOYEE WITHIN CONSISTS_OF.

END PROGRAM PROMOTION-UPDATE.

IDENTIFICATION DIVISION.
PROGRAM-ID. PERSONNEL-REPORT.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PERSONNEL-REPORT-FILE ASSIGN TO "TT:".

DATA DIVISION.
FILE SECTION.
FD  PERSONNEL-REPORT-FILE
    VALUE OF ID IS "PERSONNEL.LIS"
    REPORT IS PERSONNEL-LISTING.
```

```
WORKING-STORAGE SECTION.
01  CONTROL-FIELDS.
    02  MANAGER-NAME      PIC X(20).
    02  MANAGES-GROUP     PIC XX.
    02  SUPERVISOR-NAME   PIC X(20).
    02  SUPERVISES-GROUP  PIC XX.
    02  EMPLOYEE-NUMBER   PIC XXXXX.
    02  EMPLOYEE-NAME     PIC X(20).
REPORT SECTION.
RD  PERSONNEL-LISTING
    PAGE LIMIT IS 66
    HEADING       1
    FIRST DETAIL  3
    LAST DETAIL   60
    CONTROLS ARE  MANAGES-GROUP
                  SUPERVISES-GROUP.
01  TYPE IS PAGE HEADING.
    02  LINE 1 COLUMN 22
               PIC X(16) VALUE "EMPLOYEE LISTING".
01  MANAGER-CONTROL TYPE IS CONTROL HEADING MANAGES-GROUP.
    02  LINE IS PLUS 1.
        03  COLUMN 16 PIC X(17)
                       VALUE "MANAGER OF GROUP ".
        03  COLUMN 33 PIC XX
                       SOURCE MANAGES-GROUP.
        03  COLUMN 35 PIC XXXX
                       VALUE "IS: ".
        03  COLUMN 39 PIC X(20)
                       SOURCE MANAGER-NAME.
01  GROUP-CONTROL TYPE IS CONTROL HEADING SUPERVISES-GROUP.
    02  LINE IS PLUS 1.
        03  COLUMN 3  PIC XXXXXXX
                       VALUE "GROUP: ".
        03  COLUMN 10 PIC XX
                       SOURCE SUPERVISES-GROUP.
    02  LINE IS PLUS 1.
        03  COLUMN 3  PIC X(15)
                       VALUE IS "SUPERVISOR IS: ".
        03  COLUMN 18 PIC X(20)
                       SOURCE IS SUPERVISOR-NAME.
    02  LINE IS PLUS 2.
        03  COLUMN 3  PIC X(6)
                       VALUE "GROUP ".
        03  COLUMN 9  PIC XX
                       SOURCE IS SUPERVISES-GROUP.
        03  COLUMN 12 PIC X(9)
                       VALUE "EMPLOYEES".
        03  COLUMN 24 PIC X(15)
                       VALUE "EMPLOYEE NUMBER".
        03  COLUMN 43 PIC X(13)
                       VALUE "EMPLOYEE NAME".
01  EMPLOYEE-LINE TYPE IS DETAIL.
    02  LINE IS PLUS 1.
        03  COLUMN 28 PIC XXXXX SOURCE IS EMPLOYEE-NUMBER.
        03  COLUMN 44 PIC X(20) SOURCE IS EMPLOYEE-NAME.

PROCEDURE DIVISION.
```

```
A000-BEGIN.
    OPEN OUTPUT PERSONNEL-REPORT-FILE.
    INITIATE PERSONNEL-LISTING.
    PERFORM A100-GET-THE-BOSS THROUGH A700-DONE-THE-BOSS.
    TERMINATE PERSONNEL-LISTING.
    CLOSE PERSONNEL-REPORT-FILE.
    EXIT PROGRAM.

A100-GET-THE-BOSS.
    MOVE 10500 TO EMP_ID.
    FETCH FIRST EMPLOYEE USING EMP_ID.
    MOVE EMP_LAST_NAME TO MANAGER-NAME.
    FETCH FIRST WK_GROUP WITHIN MANAGES.
    MOVE GROUP_NAME TO MANAGES-GROUP.

A200-GET-SUPERVISORS.
    FETCH NEXT EMPLOYEE WITHIN CONSISTS_OF
               AT END GO TO A700-DONE-THE-BOSS.
    MOVE EMP_LAST_NAME TO SUPERVISOR-NAME.
    KEEP CURRENT USING KEEPSUPER.
    FETCH NEXT WK_GROUP WITHIN MANAGES.
    MOVE GROUP_NAME TO SUPERVISES-GROUP.
    PERFORM A500-GET-EMPLOYEES THROUGH A600-DONE-EMPLOYEES.
    GO TO A200-GET-SUPERVISORS.

A500-GET-EMPLOYEES.
    FETCH NEXT EMPLOYEE WITHIN CONSISTS_OF
               AT END GO TO A510-FIND-CURRENT-SUPER.
    MOVE EMP_LAST_NAME TO EMPLOYEE-NAME.
    MOVE EMP_ID        TO EMPLOYEE-NUMBER.
    GENERATE EMPLOYEE-LINE.
    GO TO A500-GET-EMPLOYEES.

A510-FIND-CURRENT-SUPER.
    FIND FIRST WITHIN KEEPSUPER.
    FREE ALL FROM KEEPSUPER.

A600-DONE-EMPLOYEES.
    EXIT.

A700-DONE-THE-BOSS.
    EXIT.

END PROGRAM PERSONNEL-REPORT.

END PROGRAM PERSONNEL-UPDATE.
```

8.7.1. PERSONNEL-UPDATE Sample Run — Listing Before Promotion

This sample report (Example 8.7), created by the preceding PERSONNEL-UPDATE program, corresponds to the data in Figure 5.24.

Example 8.7. Sample Run of PERSONNEL-UPDATE Before Promotion

```
                    EMPLOYEE LISTING
```

```
            MANAGER OF GROUP A IS: HOWELL
GROUP B2
SUPERVISOR IS: MOORE

GROUP B2 EMPLOYEES   EMPLOYEE NUMBER    EMPLOYEE NAME
                         05500           BAKER
                         02000           DEANE
                         07400           FIFER
                         05000           KLEIN
                         01400           RILEY
GROUP B1
SUPERVISOR IS: NOYCE

GROUP B1 EMPLOYEES   EMPLOYEE NUMBER    EMPLOYEE NAME
                         04000           BURLEW
                         07000           NEILS
                         01000           RAVAN
```

8.7.2. PERSONNEL-UPDATE Sample Run — Listing After Promotion

This sample report (Example 8.8, created by PERSONNEL-UPDATE in Section 8.7, corresponds to the data in Figure 5.25.

Example 8.8. Sample Run of PERSONNEL-UPDATE After Promotion

```
                  EMPLOYEE LISTING

            MANAGER OF GROUP A IS: HOWELL
GROUP B3
SUPERVISOR IS: KLEIN

GROUP B3 EMPLOYEES   EMPLOYEE NUMBER    EMPLOYEE NAME
                         07000           NEILS
                         01400           RILEY
GROUP B2
SUPERVISOR IS: MOORE
GROUP B2 EMPLOYEES   EMPLOYEE NUMBER    EMPLOYEE NAME
                         05500           BAKER
                         02000           DEANE
                         07400           FIFER
GROUP B1
SUPERVISOR IS: NOYCE

GROUP B1 EMPLOYEES   EMPLOYEE NUMBER    EMPLOYEE NAME
                         04000           BURLEW
                         01000           RAVAN
```

Appendix A. COBOL Database Programming Reserved Words

The italicized words in this list are relevant to (both) database programs *and* other programs.

ALL
ALSO
ANY

BATCH

COMMIT
CONCURRENT
CONNECT
CONTAIN
CONTAINS
CURRENCY
CURRENT

DB
DB-ACCESS-CONTROL-KEY
DB-CONDITION
DB-CURRENT-RECORD-ID
DB-CURRENT-RECORD-NAME
DB-EXCEPTION
DBKEY
DB-KEY
DB-RECORD-NAME
DB-SET-NAME
DB-STATUS
DB-UWA
DISCONNECT
DUPLICATE
DUPLICATES

EMPTY
END-COMMIT
END-CONNECT
END-DISCONNECT
END-FETCH
END-FIND
END-FREE
END-GET
END-KEEP
END-MODIFY
END-READY
END-RECONNECT
END-ROLLBACK
END-STORE
ERROR
EXCLUSIVE

FETCH
FIND
FIRST
FREE

GET

KEEP

LAST
LD
LIMIT
LIMITS

MATCH
MATCHES
MEMBER
MEMBERSHIP
MODIFY

NEXT
NULL

OFFSET
OTHER
OTHERS
OWNER

PRIOR
PROTECTED

READY
REALM
REALMS
RECONNECT
RECORD
RELATIVE
RETAINING
ROLLBACK

SET
SETS
STORE
SUB-SCHEMA

TENANT

UPDATE
UPDATERS
USING

WAIT
WHERE
WITHIN

Glossary of Oracle DBMS-Related Terms

access mode
In a database environment, that part of the COBOL data manipulation language READY statement's usage mode that describes what capabilities your run unit will have with regard to records in the realm you have readied. The access mode can be RETRIEVAL (read only) or UPDATE (read and write).
See Also **usage mode**, **allow mode**.

allow mode
That part of the DML READY statement's usage mode that describes what you will allow other run units to do while your run unit works with storage areas in the realms you have readied. The allow mode can be CONCURRENT, PROTECTED, EXCLUSIVE, or BATCH.
See Also **usage mode**, **access mode**.

at end condition
A condition caused during FETCH or FIND statement execution for a database, when no next logical record exists.

AUTOMATIC member
A database record that automatically becomes a member of a given set when the record is stored in the database. AUTOMATIC set membership is declared in the schema.

available mode
The state of a database record that allows its use by the Database Control System (DBCS) in executing an operation requested by a given run unit. A record is available if it is stored in a database area accessible to the DBCS, and the intended use does not conflict with the processing requirements of concurrent run units.

Bachman diagram
A graphic representation of the set relationships between owner and member record types used to analyze and document a database design.

BATCH RETRIEVAL usage mode
The state of a realm which allows concurrent run units to update the realm while the current run unit accesses a copy of the realm which was made at the point when the READY was executed.

BATCH UPDATE usage mode
The state of a realm in which the current run unit may access or update any data in the realm while allowing concurrent run units to retrieve from the realm but preventing them from updating the realm. Effectively similar to PROTECTED UPDATE.

CDD/Repository
See **Oracle CDD/Repository**.

CODASYL
An acronym for the COnference on DAta SYstems Languages, the committee that produced the document titled *CODASYL COBOL Journal of Development*. This document serves as the basis for the standardization of the Oracle CODASYL DBMS data manipulation language (DML).

CONCURRENT usage mode
The state of a realm in which it may be accessed by concurrent run units.

concurrency
The simultaneous use of a database or a sequential, relative, or indexed file by more than one user.

currency indicators
Pointers maintained by the Database Control System (DBCS) that serve as place markers in the database for your run unit.

data definition languages (DDL)
The languages used to describe schemas, subschemas, and storage schemas.
See Also **schema DDL**, **storage schema DDL**, **subschema DDL**.

data manipulation language (DML)
The Oracle CODASYL DBMS language interface that permits programs to interact with Oracle CODASYL DBMS databases.

data-name
A user-defined word that names a data item described in a data description entry. In general formats, data-name represents a word that must not be reference-modified, subscripted, indexed, or qualified unless specifically allowed by rules of the format.

database
A collection of related records on a mass storage device. All of the records and sets are controlled by a specific schema.

database administrator (DBA)
The person or group of people responsible for planning, designing, implementing, and maintaining a database.

database aggregate
A subschema group item or table defining one or more database items. A database aggregate can contain one or more database aggregates.

Database Control System (DBCS)
The component of Oracle CODASYL DBMS that, together with the OpenVMS operating systems, provides run-time control of database processing.

database exception condition
The state that exists for a run unit when the DBCS detects a situation for that run unit that requires special handling.

database item
An elementary data item defined in a subschema. It corresponds uniquely to a data item in the subschema's host schema.

database key (dbkey)
A numeric value that uniquely identifies a record in the database. The Database Control System assigns the value when a record is stored in the database. Although your run unit cannot directly access database keys, they are used by the Database Control System whenever you store, retrieve, or manipulate a record. Dbkey values are notated in the form *x:y:z*, where *x* is the area, *y* is the page, and *z* is the record number.

database key condition
A condition for which a truth value can be determined, that: (1) two specified database key values identify the same database record, (2) a database key value is null, (3) or a key value is identical to any database key value in a keeplist.

database key identifier
A phrase in a COBOL source program that refers to a database key value within a currency indicator or a keeplist.

database management system (Oracle CODASYL DBMS)
A system for creating, maintaining, and accessing a collection of interrelated database records that may be processed by one or more applications without regard to physical storage. Oracle CODASYL

	DBMS establishes logical relationships among records. Data is described independently of application programs, providing ease in application development, data security, and data visibility. Oracle CODASYL DBMS is available under a separate license.
database object	A set type, record type, realm, record key, or data item defined in the schema.
database page	The unit of data transfer between Oracle CODASYL DBMS and the OpenVMS operating systems. Each database page consists of one or more blocks of 512 bytes each.
Database Query utility (DBQ)	An online interactive utility that allows the user to access a Oracle CODASYL DBMS database directly and that shows the results of each execution of a DML statement. This utility provides low-level query facilities for data processors.
DB-CONDITION	A database special register whose value indicates either a successful condition or an exception condition.
DB-CURRENT-RECORD-ID	A database special register containing the subschema user ID number (UID) of the record type of the current record of the run unit. It contains zero if there is no current record of the run unit.
DB-CURRENT-RECORD-NAME	A database special register containing the name of the record type of the current record of the run unit. It contains spaces if there is no current record of the run unit.
DBA	See **database administrator (DBA)**.
DBCS	See **Database Control System (DBCS)**.
dbkey	See **database key (dbkey)**.
DB-KEY	A database special register that holds the dbkey of the record accessed by the last FETCH, STORE, or FIND statement. This special register can be used to fine tune storage strategies in the database.
DBMS	See **database management system (Oracle CODASYL DBMS)**.
DBQ	See **Database Query utility (DBQ)**.
DB-UWA	A database special register which serves as the record delivery area that the Database Control System (DBCS) uses to make data items available to your program.
DDL	See **data definition languages (DDL)**.
de-edit	The logical removal of all editing characters from a numeric edited data item in order to determine that item's unedited numeric value.
deadlock	A database processing situation in which two or more run units are stopped by conflicting requests for locked records.
DML	See **data manipulation language (DML)**.
DMU	Dictionary Management Utility.

empty set
A database set occurrence containing no member records.

EXCLUSIVE usage mode
The state of a realm in which it cannot be accessed by a concurrent run unit.

FIXED member
A record, upon becoming a member of a set occurrence of a FIXED set type, that must remain a member of that set until it is erased from the database. Fixed set membership is declared in the schema DDL. Compare with **MANDATORY member**.

INSERTION class
An attribute of member records that describes how and when members are added to database sets.
See Also **AUTOMATIC member**, **MANUAL member**.

journal file
A database file that contains all records modified by a run unit, usually chronologically ordered. A journal file allows reconstruction of the data to prefailure conditions in case of database contamination due to system or program failures.

journaling
The act of creating, writing, or both, to a journal file.

junction record
A record inserted between two records of the same type. You can use a junction record to simulate what would otherwise be an illegal set relationship for Oracle CODASYL DBMS, that is, a record type being a member of a set that it owns. Also, using a junction record helps avoid data redundancy and inconsistency.

keeplist
A list of database keys used by a run unit to lock records for later reference.

locking
VSI COBOL for OpenVMS has facilities that allow concurrent use of a database or a sequential, relative, or indexed file without corrupting their records. RMS on OpenVMS VAX maintains locks on a file, whereas RMS on OpenVMS maintains locks on files and records. In Oracle CODASYL DBMS, locks are maintained on individual records, entire realms, or both.

MANDATORY member
A record, upon becoming a member of a set occurrence of a set type, must remain a member of that or some other set occurrence of that set type until it is erased from the database. MANDATORY set membership is declared in the schema. Compare with **FIXED member**.

MANUAL member
A database record that becomes a member of a given set by explicit direction of the application program using the CONNECT statement. MANUAL set membership is declared in the schema.

member condition
The condition, for which a truth value can be determined, that a database record is a member of one or more sets.

member record
A database record, other than the owner record, included in the set. There may be one or more member record types in a set. There may be zero or more member records in a set.

nonsingular set
A database set not owned by the SYSTEM.

See Also **SYSTEM-owned set**, **owner record**.

null

A data attribute associated with currency indicators and database key values. This attribute is independent of the value of the contents of data items.

OPTIONAL member

A database record that does not necessarily remain a permanent member of a set. Its membership in a set may be changed using the DISCONNECT statement without its being deleted from the database. OPTIONAL set membership is declared in the schema.

Oracle CDD/Repository

The central repository of information about data elements, data structures, and relationships between data structures. Oracle CDD/Repository is used by Oracle CODASYL DBMS, Datatrieve, and VSI COBOL for OpenVMS. It does not contain actual data files. Rather, it contains definitions of schemas, storage schemas, and subschemas. Oracle CDD/Repository is available under a separate license.

owner record

The head of a group of database records that make up a set. There can be only one record type as the owner for each set type and one owner record occurrence for each set occurrence.

PROTECTED usage mode

The state of a realm in which it may be retrieved from but cannot be modified by concurrent run units.

quiet point

A time when no run units are accessing a database. Quiet points and transactions are mutually exclusive (for the entire database). Compare with transaction.

For the run unit, the time between a COMMIT or ROLLBACK, and the following READY.

ready mode

The state of a realm after execution of a READY statement for that realm and before the execution of a COMMIT or ROLLBACK statement for that realm.

realm

One or more schema areas. Realms are declared in the subschema.

realm currency indicator

A currency indicator (in other words, database key value) associated with a particular realm. A realm currency indicator identifies a particular database record, position in the realm, or both.

record key

A key whose contents identify a record in an indexed file or within a record type in a database. Within an indexed file, record key is either the prime record key or an alternate record key. Within a database, a record key may or may not have ordering significance.

record occurrence

A user-stored instance of a record type. A record occurrence is the actual physical data representation of a single record in the database, but not its definition, which is the record type.

record order key

A record key associated with a record type in a database. The definition of a record order key in the schema causes the DBCS to maintain the records in the specified logical sequence based on values of the record order key.

record selection expression A word or group of contiguous words in a COBOL source program that specifies the algorithm to be used by the Database Control System (DBCS) to identify a specific database record.

record type currency indicator A currency indicator associated with a particular database record type. A record type currency indicator identifies a particular record of the record type. If the subschema includes a record order key for a record type, its associated currency indicator identifies a particular record, position in the record type, or both.

RETRIEVAL usage mode The state of a realm in which the current run-unit may only retrieve from it.

schema The logical description of a database, including data definitions and data relationships. The schema is written using the schema data definition language (schema DDL).

schema DDL The language used to define the logical structure of a database.

schema-name A user-defined word that identifies a schema.

section header A combination of words (followed by a separator period) that indicates the beginning of a section in the Environment, Data, and Procedure Divisions of a COBOL program. In the Environment and Data Divisions, a section header consists of reserved words followed by a separator period in the Division. SUB-SCHEMA SECTION is a valid section header.

set A defined relationship among records in a database. A set contains an owner record and zero or more member records.
See Also **set occurrence**, **set type**, **empty set**.

set member A record stored in the database as a nonowner participant in a specific set.

set-name A user-defined word that identifies a set type.

set occurrence An instance of a database set type. A set occurrence is the actual data in the set, not its definition, which is the set type.

set-ordering criteria The specification for the positioning of a member record in a set by the DBCS. The schema defines this specification.

set owner A database record occurrence whose existence establishes the existence of a specific set occurrence.

set type A specific named set that has been defined in the schema data definition language. It is the definition of a collection of sets that have identical characteristics. Set types are declared by the schema data definition language.

set type currency indicator A currency indicator associated with a particular database set type. A set type currency indicator identifies a particular set of the set type and a particular record, position in that set, or both.

simple condition Any single condition from the following list:

relation condition
class condition
condition-name condition
switch-status condition
sign condition
success/failure condition
simple-condition (in parentheses)
tenancy condition (Oracle CODASYL DBMS simple condition type)
empty condition (Oracle CODASYL DBMS simple condition type)
database key condition (Oracle CODASYL DBMS simple condition type)

singular set
See **SYSTEM-owned set**.

storage schema
A description of the physical storage of data in database files. The storage schema is written using the storage schema data definition language.

storage schema DDL
The language used to define the physical organization of the database.

SUB-SCHEMA SECTION
The section of the Data Division that defines the subschema and keeplists to be used by the COBOL program.

subschema
A user view of a database. The subschema can include everything in the original schema DDL or any part thereof. The subschema is written using the subschema data definition language (subschema DDL).

subschema DDL
The language used to define the user view of a database.

subschema entry
An entry in the Subschema Section of the Data Division that specifies the subschema to be accessed by the COBOL program.

subschema-name
A user-defined word or nonnumeric literal that identifies a subschema.

SYSTEM-owned set
A set owned by the SYSTEM rather than by a record type. SYSTEM-owned sets have only one occurrence in the database and are used for relationships with large numbers of member occurrences or as entry points into the database.

tenancy condition
The condition, for which a truth value can be determined, that a record is a member, owner, or either, of one or more sets.

tenant
A database record that is either the owner or a member of a specific set.

tenant record
An owner or member record of a set.

UPDATE usage mode
The state of a realm in which the current run-unit may update it, as well as retrieve from it.

usage mode
The combination of the DML READY statement's allow mode and the access mode. The usage mode describes how a READY realm can be accessed. The eight usage mode combinations are:

CONCURRENT RETRIEVAL

CONCURRENT UPDATE

EXCLUSIVE RETRIEVAL
EXCLUSIVE UPDATE

PROTECTED RETRIEVAL
PROTECTED UPDATE

BATCH RETRIEVAL
BATCH UPDATE

PROTECTED and RETRIEVAL are the default.
See Also **access mode**, **allow mode**.

user work area (UWA)

A portion of memory assigned to the program's run unit at run time. The run unit delivers data for the DBCS to this area, and it is here the DBCS places data requested from the database for retrieval by the run unit.