Full IDS/II

User's Guide

DPS7000/XTA
NOVASCALE 7000

Database Products: Full IDS-II

REFERENCE
47 A2 07UDA00

DPS7000/XTA
NOVASCALE 7000
Full IDS/II

User's Guide

Database Products: Full IDS-II

Septembre 1991

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 07UDA00

Trademarks and Acknowledgements

Preface

SCOPE AND OBJECTIVES

This manual describes the overall organization of the IDS/II database model. It describes the IDS/II database form two points of view: from the point of view of a static or conceptual database, and from the point of view of a real, dynamic database. The discussions based on a conceptual database are used to explain the theoretical aspects of the IDS/II product. Discussions on the dynamic database demonstrate how the database actually works in terms of the need for constant modifications on the contents in the database.

This manual does not discuss the processors which are required for the operation of IDS/II.

INTENDED READERS

This user guide is to be used by persons having the responsibility of organizing or administering an IDS/II database. Those persons can refer to this manual for information on the concepts of organization and operation of the IDS/II database and for explanations on how to manage the contents of the database.

STRUCTURE OF THIS DOCUMENT

This manual is divided into eight sections.

Section I is a general introduction to the structure of the IDS/II database.

Section II discusses IDS/II from the point of view of a static database.

Sections III to VII discuss the dynamic IDS/II database. Please refer to Section I for a further explanation of which aspects of IDS/II are discussed in each of these sections.

Section VIII gives complementary notions on the IDS/II model. It discusses concurrent access to the same database by several run-units, communication between programs and run-time consistency.

RELATED DOCUMENTS

The manuals below provide the information necessary for the use of an IDS/II database in the Version 5 environment.

Full IDS/II Reference Manual Volume 1	*47 A2 05UD*
Full IDS/II Reference Manual Volume 2	*47 A2 06UD*
Full IDS/II Administrator's Guide	*47 A2 13UD*
Database Reorganization Utility User's Guide	*47 A2 15UD*

SYNTAX NOTATION

The notation used in this manual, and the rules that apply to those formats are listed below :

- The elements that make up a clause are: upper case words, lower case words, special symbols, and special characters.

- All underlined upper case words (keywords) are required when formats are used.
- Example: <u>SCHEMA</u> NAME IS schema-name.

- Upper-case words which are not underlined are optional words; in the preceding example, NAME and IS are optional.

- Lower-case words are generic terms that must be replaced by appropriate names or values. In the preceding example, schema-name is a generic term and must be replaced by the user-defined name for the schema.

- When a portion of a general format is enclosed in spacial symbols, the followuing rules apply:

1) Brackets indicate that the choice of a parameter is optional: no selection is necessary.

   [a]
   [b]
   [c]

2) Braces indicate that the choice of one parameter is required: one parameter must be selected, and only one.

   {a}
   {b}
   {c}

3) Double bars indicate that the choice of at least one parameter is required: at least one parameter must be selected; atmost an occurrence of each parameter can be selected.

   |a|
   |b|
   |c|

An ellipsis (...) indicates that repetition is allowed. The portion of the format that can be repeated is determined by the bracket ([) brace ({) which logically matches the bracket (]) or brace (}) to the immediate left of the ellipsis.

Table of Contents

Illustrations

Figures

Tables

1. Introduction

IDS/II is a data management subsystem that provides the user with:

- **A structural model**, based on records and relationships between those records.
- **A methodology for the use of this model**, characterized by the separation between the description of the data structure of the model and the COBOL programs that deal with it.

1.1 STRUCTURAL MODEL BASED ON RECORDS AND SETS

In a typical COBOL application program, the entity processed is the **record**. Records are grouped by type into sequential, indexed sequential or direct files. In the example given in Figure 1-1, CUSTOMER records and PART records are placed in indexed sequential files, ORDERS records are placed in an unsorted sequential file. (The name ORDERS is used in the plural because ORDER is a reserved word in the languages described later.)

If the need arises to list all the orders of a given customer or all the orders involving a given part, it is the responsibility of the programmer to search the entire ORDERS file for orders concerning that customer or that part. In IDS/II, these (implicit) relationships are usually established by placing fields in the ORDERS record which contain the keys of associated CUSTOMER and PART records.

In an IDS/II environment, these relationships (called **sets**) become an explicit part of the data structure, just like the **records**. Each set is given a specific name, for example ORDERS-OF-CUSTOMER or ORDERS-WHERE-PART, and the relationships defined by that set are maintained entirely by IDS/II, usually in the form of a chain of pointers.

As for the records themselves, they are located in "multi-record-type" files, which constitute the **database**.

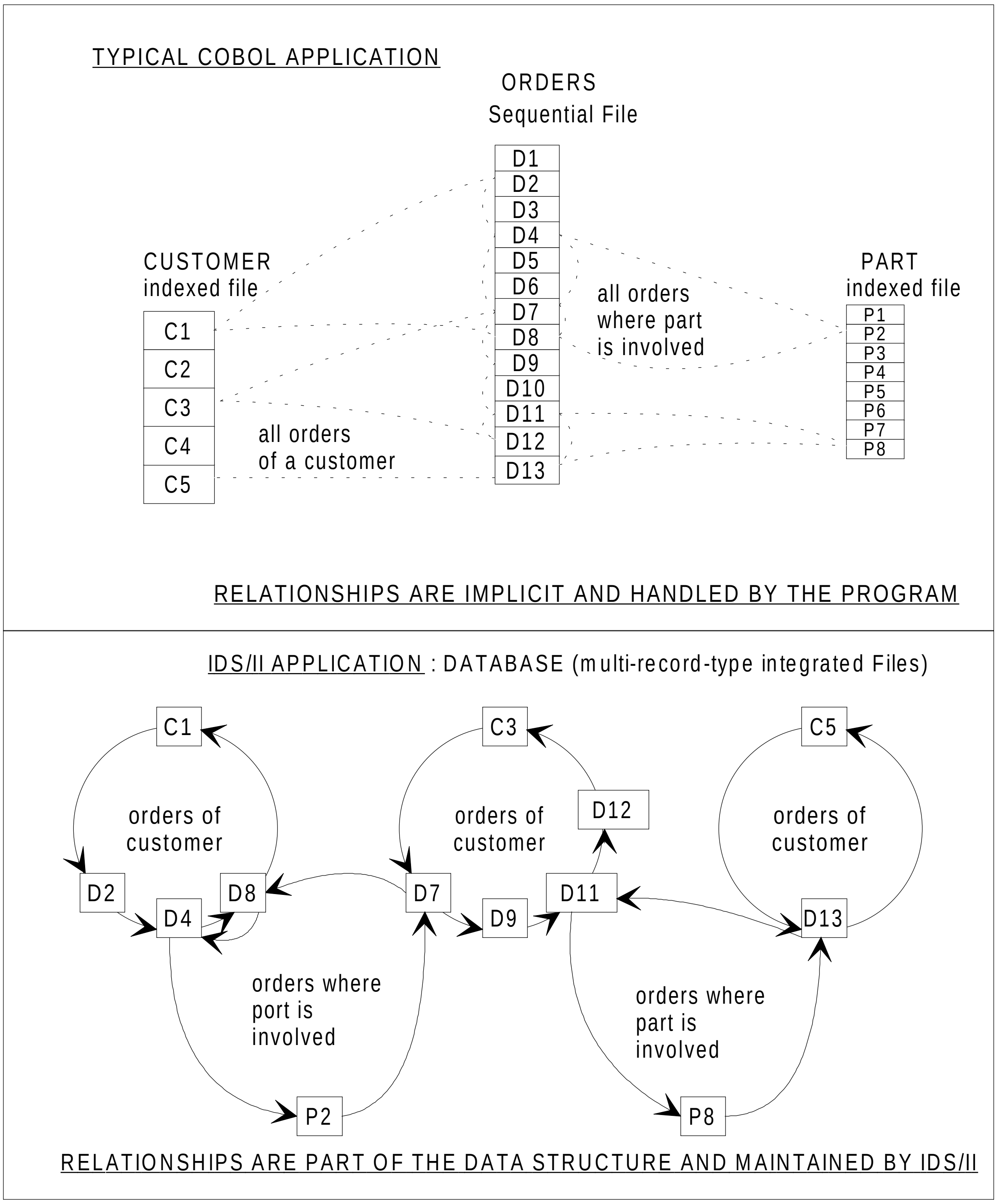


Figure 1-1. Relationships Among Records

In order to find a customer's orders, the programmer must use a sequence of statements in the following form:

```
    FIND ANY CUSTOMER.    (Direct access by key)
 LOOP.FIND NEXT ORDERS WITHIN ORDERS-OF-CUSTOMER.
                                (Sequential access along a set)

      ...
      GET ORDERS.
      ...
      GO TO LOOP.
```

When an ORDERS record is called by the program from a sequential file, its **customer key** and **part key** fields serve to identify the related CUSTOMER and PART records, which are already in the database. But when the ORDERS record itself is stored in the database, these fields become redundant and can be removed. They are in fact replaced by the **set pointers**.

This removes data reduncancy in an IDS/II database with the following advantages:

- in update mode, maintaining consistency among several copies of the same information no longer exists;
- less space is taken up on the storage media (disk)

1.2 METHODOLOGY

The basic method used for manipulating an IDS/II database is to describe the structure of the database **only once** and to use this centralized description as the unique reference for all applications which process the information contained in the database (see Figure 1-2).

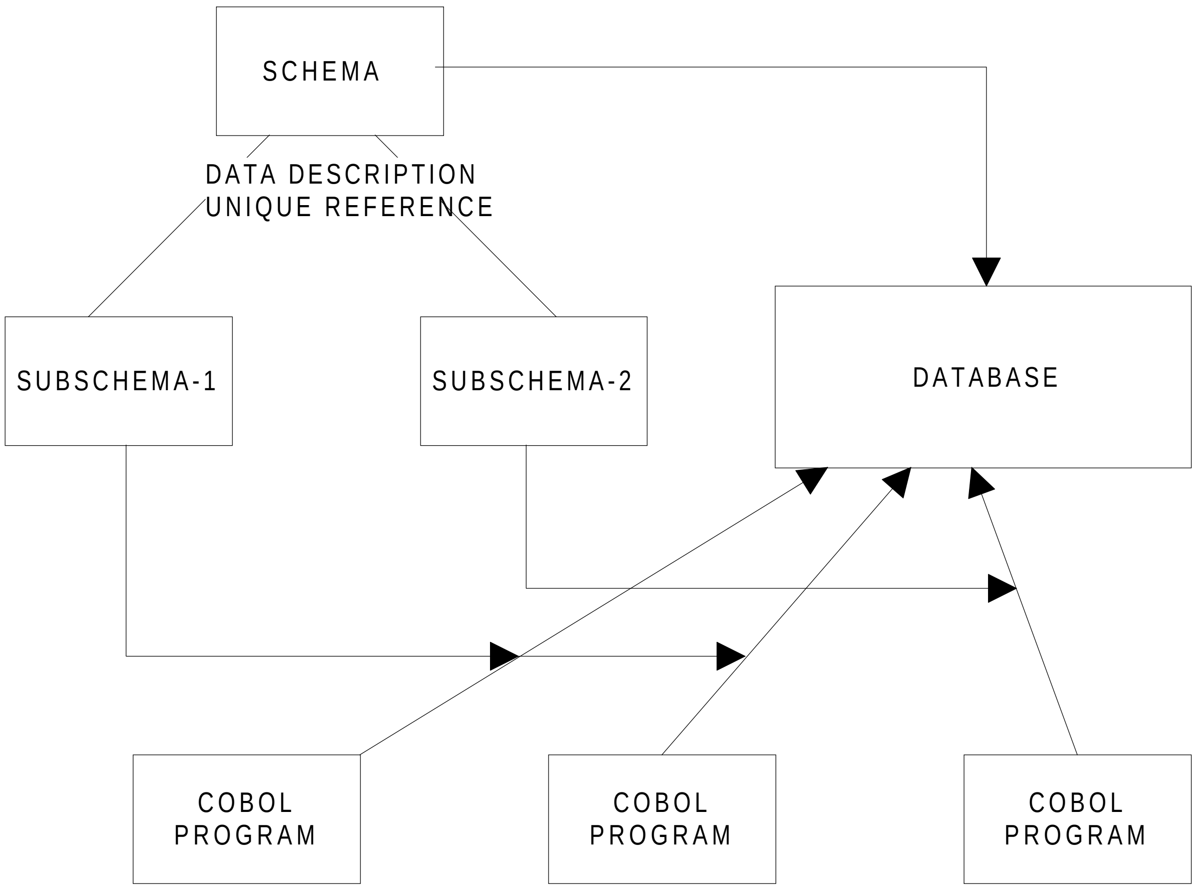


Figure 1-2. Separation of Data Structure Description and Database Manipulation

1.2.1 Data Description

- The **schema** is the definition of the structure of the database in terms of fields, records and sets. It is written in a language called **Schema Data Description Language** (DDL) which is different from COBOL. The person responsible for the design of the schema is called the **Database Administrator** (DBA).
- A complementary language, the **Device/Media Control Language** (DMCL), enables the Database Administrator to define how the database is to be mapped onto UFAS integrated files.
  - The **subschema** describes a partial view of the database which is available to a given set of application programs. Several subschemas may be defined against the schema and each subschema may be used by several application programs. Since the subschema is a partial view of the database, it only makes available information which a given program needs. Thus, the mechanism of the subschema provides protection to the rest of the database, since unrequired information remains untouched.
  - Another important factor is that the subschema isolates a program from the parts of the database which it does not access, and therefore renders the rest of the database less sensitive to changes which are outside its scope.

1.2.2 Data Manipulation

In a COBOL application program, the database is accessed via the **Data Manipulation Language** (DML), which is an extension to the COBOL language.

DML consists of new statements which must be inserted into certain sections of the COBOL program, as described below:

- In the DATA DIVISION, the SUBSCHEMA SECTION references the subschema which is to be used in association with this program. It also performs, like a COPY statement, by making available to the program the database record descriptions which are centralized in the schema.
- In the PROCEDURE DIVISION, verbs which are adapted to the record-set model are added to the list of the COBOL file manipulation verbs. The program can thus process BFAS files (for GCOS releases up to and including GCOS7 V3), UFAS files and the database areas at the same time (see Figure 1-3).

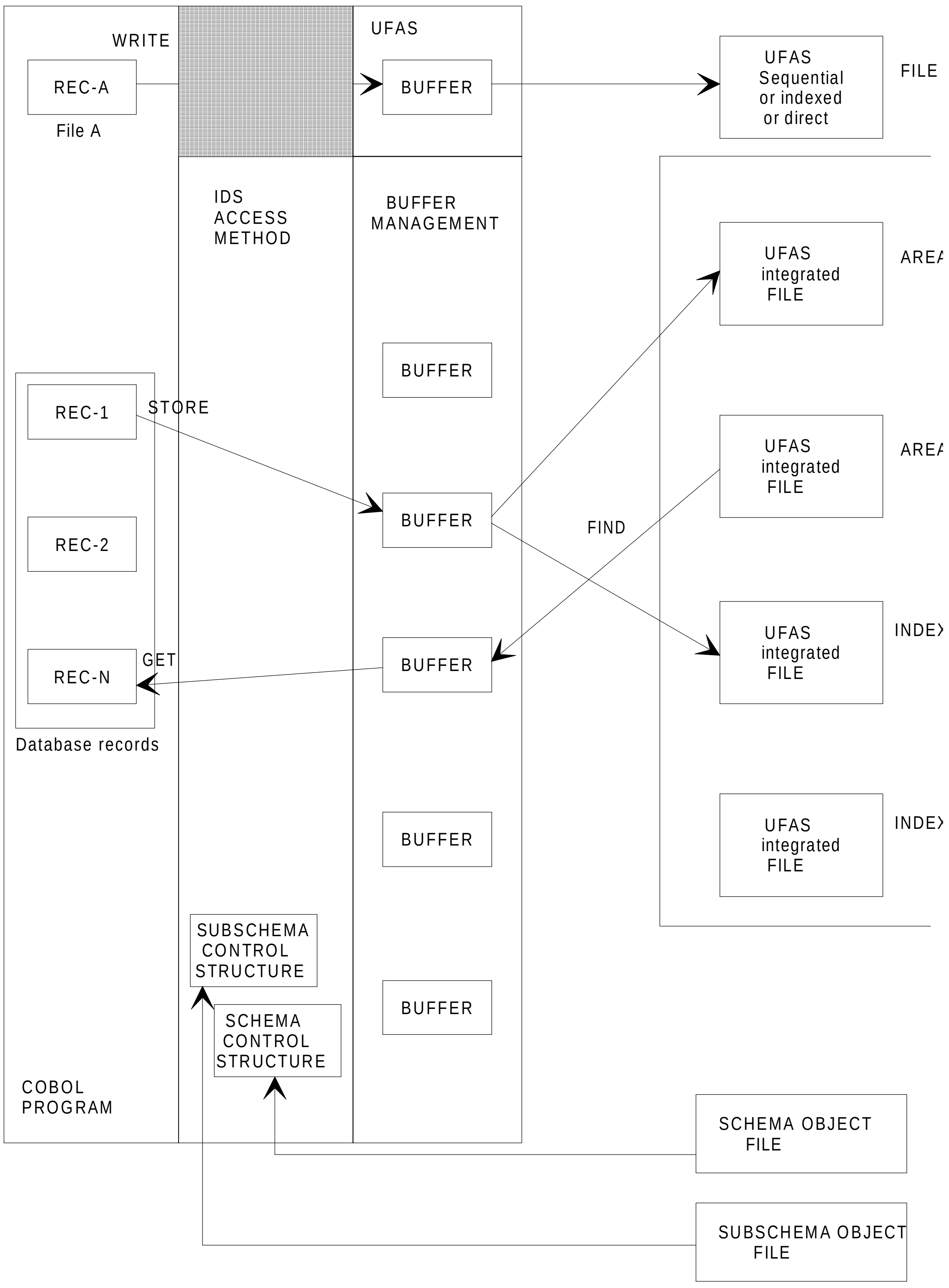


Figure 1-3. DML as an Extansion of the COBOL File Manipulation Statement

1.3 IDS/II PRODUCT OVERVIEW

The components which make up the IDS/II product are presented in Figures 1.4 through 1.9. This series of figures illustrates how these components are to be used progressively, beginning with the design of the schema and application programs up to the execution of those programs. The following paragraphs also give a step-by-step description of how to use the IDS/II product.

1) First the user must write the Schema description, by using the Data Description Language (DDL).

2) Once the Schema is completed, the user must write a Subschema description, which describes a partial view of the database. The Subschema description must be written in the Subschema Data Description Language (SDDL).

3) The Schema and Subschema descriptions are then submitted to the **DDL/SDDL/DMCL processor** by using either the JCL command DDPROC or the GCL command MNDD. That processor in turn creates an object schema file and an object subschema file which are then placed in a library.

4) The next step is to describe the physical storage characteristics of the database by using the Device/Media Control Language (DMCL). This description is then submitted to the DDL/SDDL/DMCL processor, which updates the object schema file previously created with the storage information contained in the DMCL.

5) The **PREALLOC utility**, with the information from the DMCL, will then allocate the UFAS integrated files which constitute the database. The PREALLOC utility retrieves the necessary DMCL information from the object schema file.

6) Once steps 1 through 5 are completed, application programs must be submitted to the **COBOL compiler**, which inserts the database record-descriptions into the DATA DIVISION using the information in the referenced object schema. The COBOL compiler then interprets the DML statements and compiles the program, creating a compile-unit.

7) The **linker** is then executed to create a load-module.

8) Once the load-module is created, the program step or **run-unit** is started. DML statements start the run-unit by activating the IDS/II access method, which is called the Database Control System or **DBCS** throughout the IDS/II manual set. Once the DBCS loads the schema control structure from the object schema file, it interfaces with the UFAS access method for all input-output operations on database files. The user can request a trace of DML statement activity and relevant statistics and timing information by choosing the monitored mode among the run-time options.

The following paragraphs provide a brief description of utilities and commands which are available to the IDS/II user for manipulation and control of the database.

1) Areas of the database may be saved and restored using the standard **FILSAVE** and **FILREST** utilities.

2) The database utility, **DBUTILITY**, displays information on the database and modifies its contents by means of the various commands.

3) The **ANALYSE** command displays information about an existing database.

4) The **ANALYSE INDEX** command displays information about an existing index file.

5) The **SIMULOAD** command simulates the loading of records and displays statistical information about the simulated database.

6) The **PRINT** command prints the contents of records and/or pages of the database.

7) The **VALIDATE** command performs consistenty checks on the database.

8) The **DELETE_KEY** command deletes one or several secondary keys from an index file.

9) The **BUILD_KEY** command reconstructs one or several secondary keys in an index file.

10) The **PATCH** command performs patching of local inconsistencies.

11) The **RECONNECT** command reconnects selected CALC records to their correct calc-chains, according to the values of their calc-keys.

12) The **Interactive DML** commands activate the IDS/II access method in the same way as a COBOL program.

13) The database utility, **DBREORG**, performs a physical reorganization of a database, which becomes necessary in the following three situations:

    if the logical design of the database (described by the Schema DDL) has to be modified to reflect a structural reorganization;

    if the physical design of the database (described by the Schema DMCL) has to be modified to resolve space problems on the disk;

    if the placement of records in the database has to be recalculated for better performance.

14) The **TRANSLATE** command produces an internal file, the Schema TRANSFORM file, using information from both the old and new versions of the Schema DDL/DMCL; that file contains any necessary database tranformations the user requires.

15) The **ANALYSE IMPACT** command produces a report on the impact which changes would have: a) on the database, and b) on user application programs, if the user were to reorganize the database.

16) The **REORGANIZE** command performs the entire database reorganization specified in the schema TRANSFORM file, **excluding** record migration (see LOAD/UNLOAD facility below).

17) The **LOAD/UNLOAD** facility, which is provided to resolve database performance problems, performs the entire database reorganization **including** record migration.

18) The **SAVE** command stores a copy of the contents of the database onto a sequential file which can be used for database operation during any reorganization of the database.

Figures 1.4 to 1.9, below and on the following pages, show the progressive operation of the components of an IDS/II database.

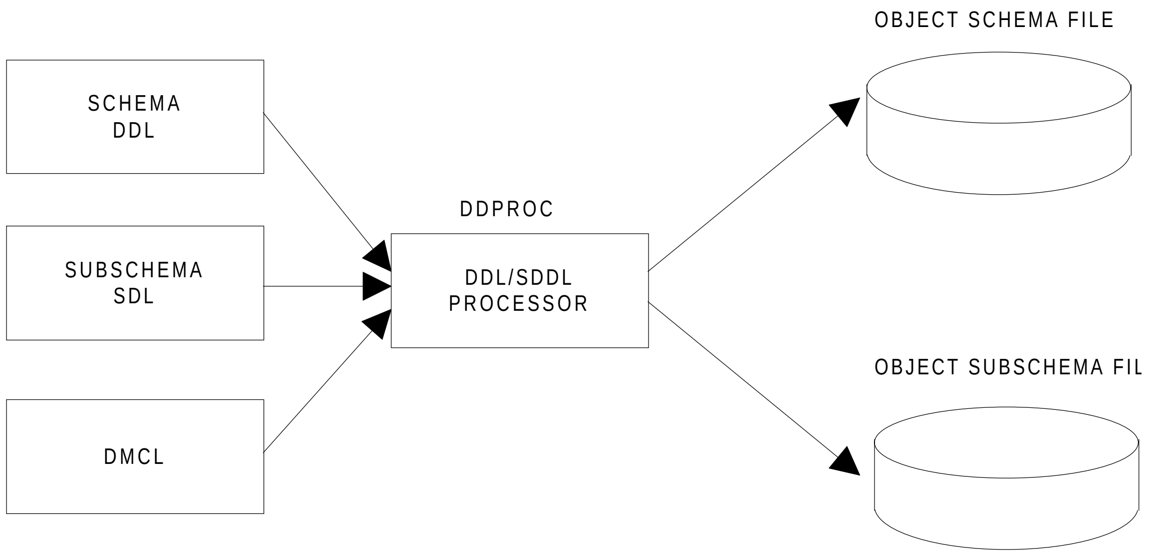


Figure 1-4. Schema Compilation Phase

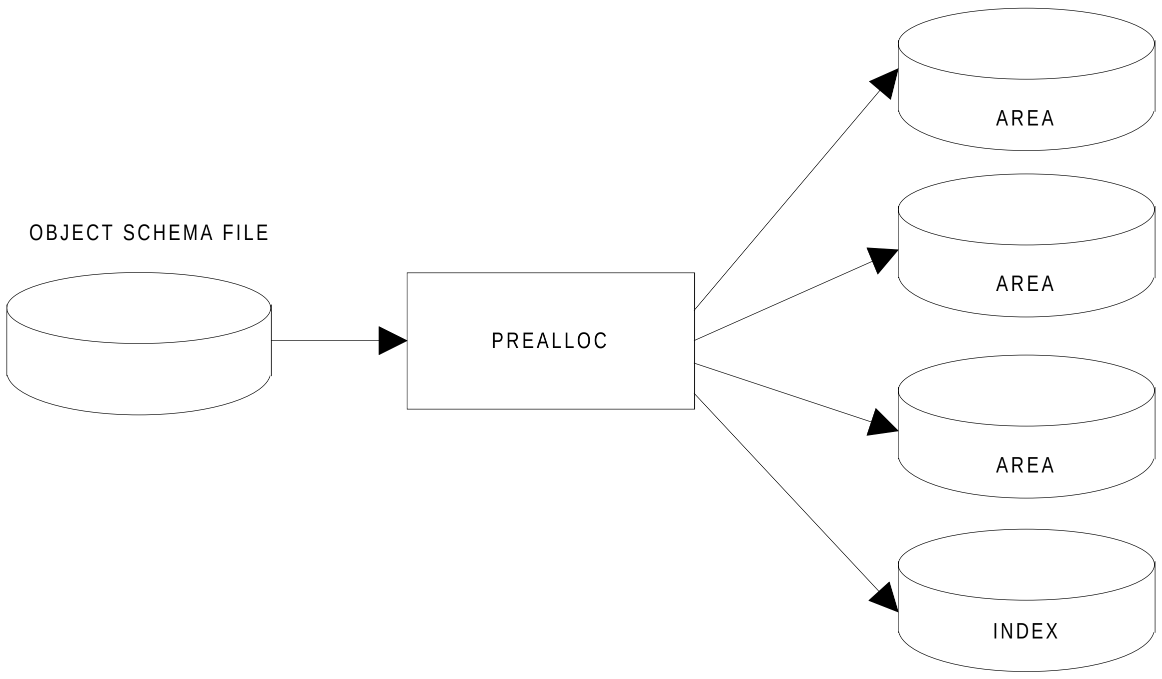


Figure 1-5. Preallocation Phase

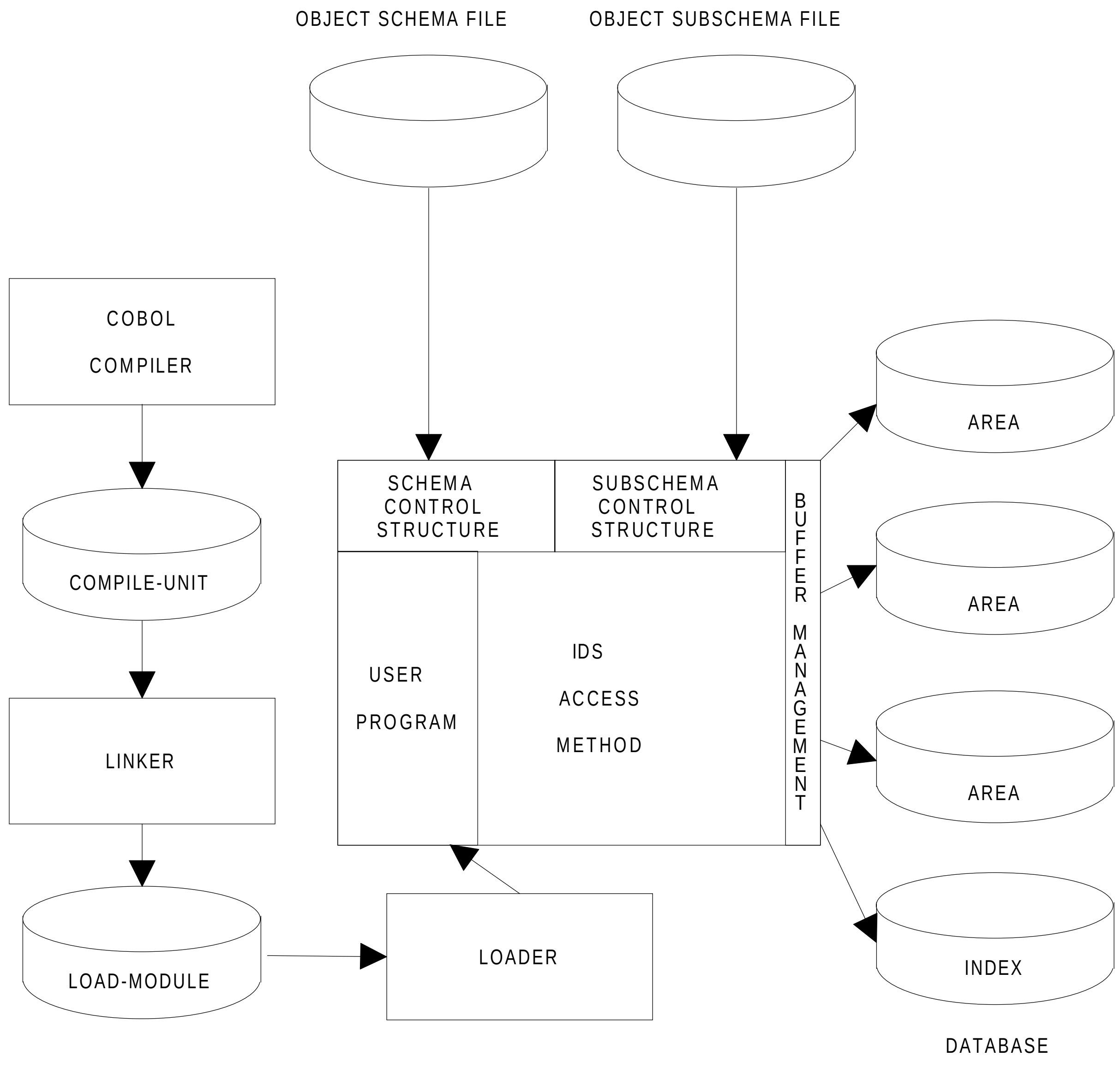


Figure 1-6. Run-time Phase

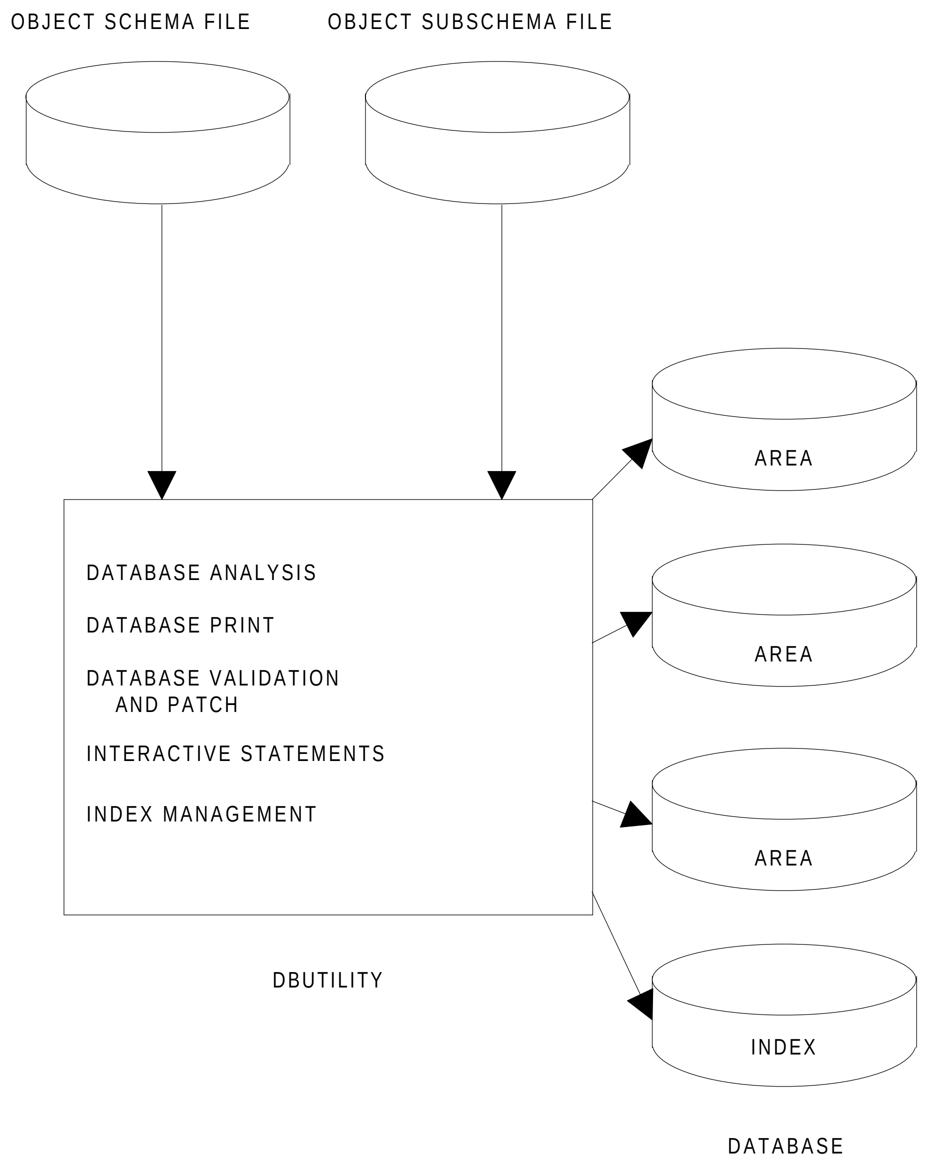


Figure 1-7. DBUTILITY Phase

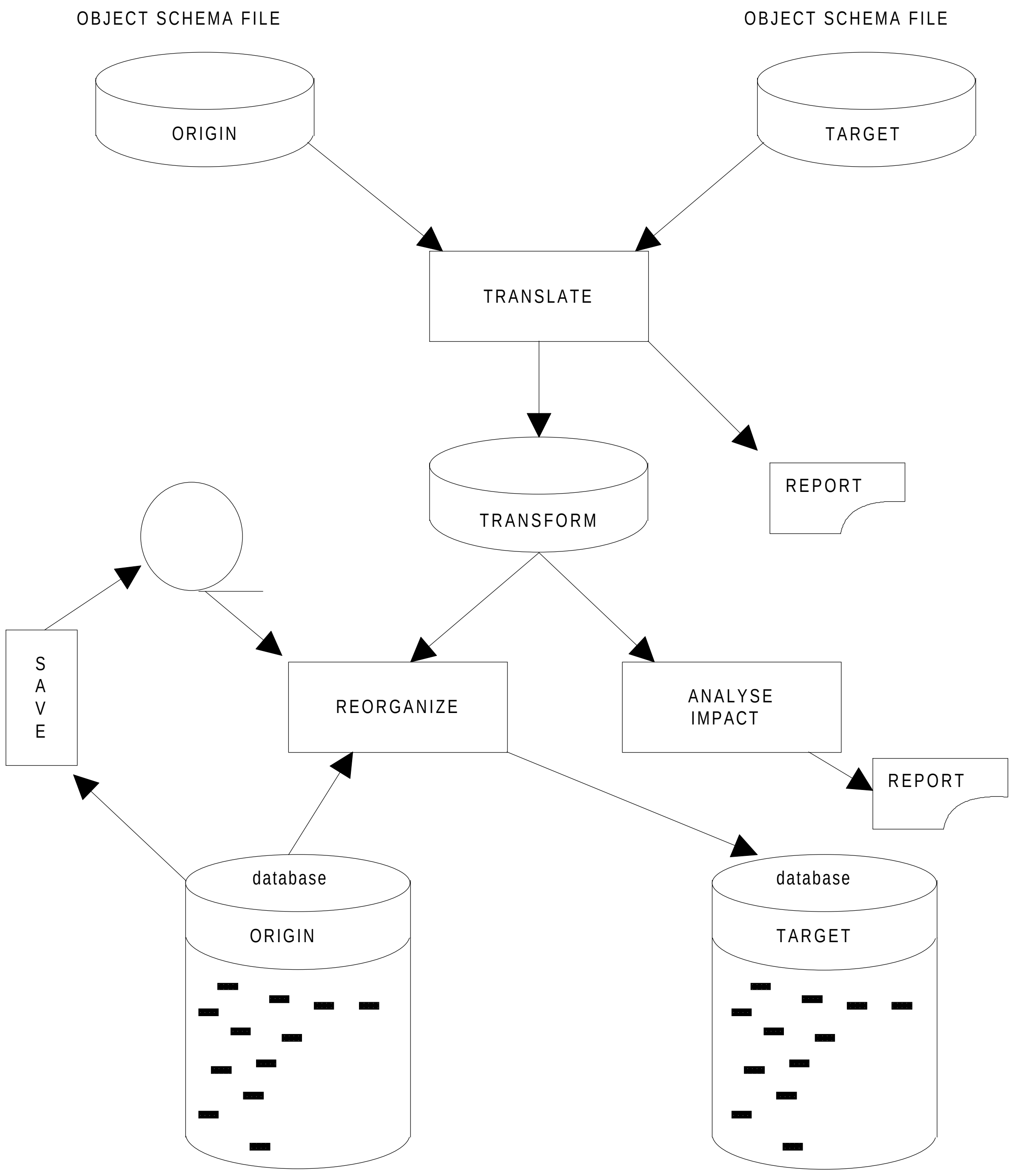


Figure 1-8. DBREORG Phase without Migration

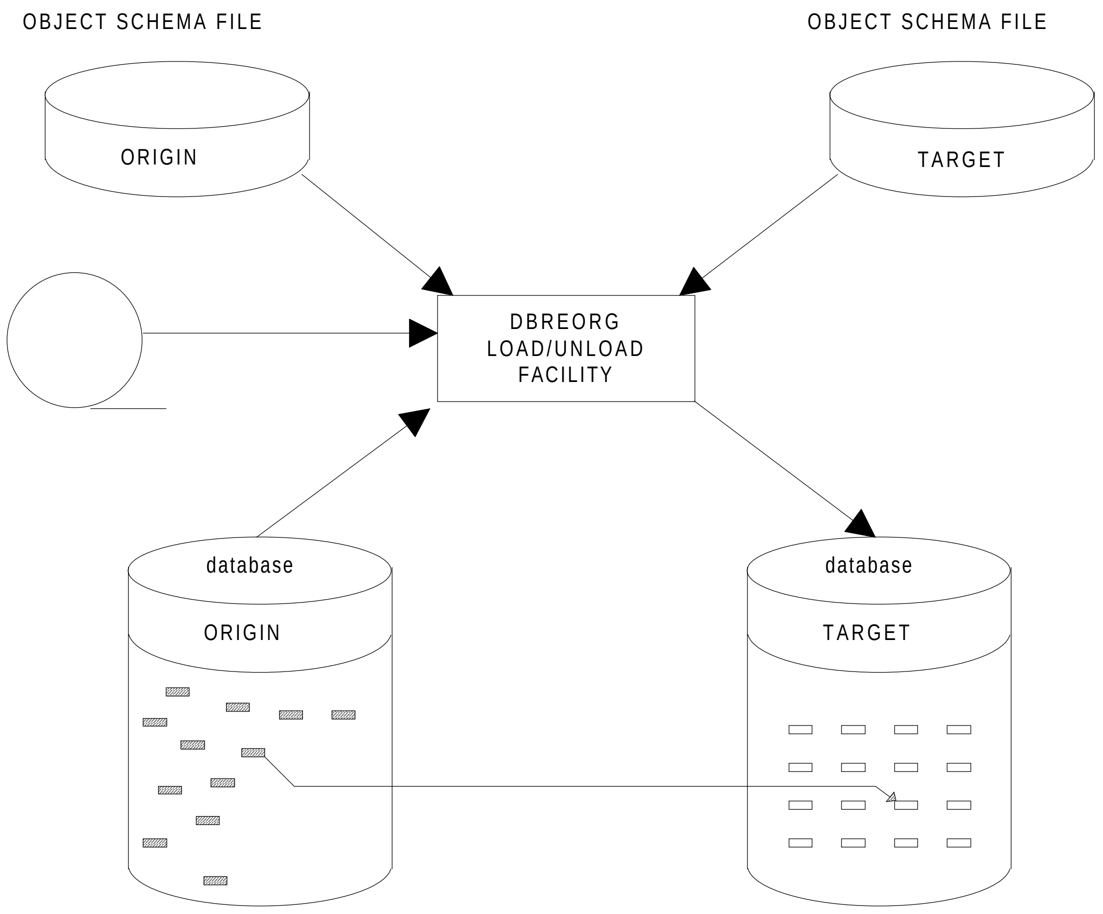


Figure 1-9. DBREORG Phase with Migration

1.4 SCOPE OF THE IDS/II USER GUIDE

The scope of this manual is limited to the overall organization of the IDS/II database model. It does not deal with the operation of the processors which are required for the operation of the IDS/II database.

1.4.1 Structure of this Manual

This manual describes the IDS/II database from two basic points of view:

- the static or conceptual database (an ideal database whose contents never need to be modified), and
- the dynamic database (a real database which constantly requires changes).

The static database concept is used to help the unexperienced user to understand the intrinsic structure of the IDS/II product in terms of **records**, **sets**, and **types**. Discussions about the static database are not concerned with **record**-**occurrences**, but only with the records themselves within the structure of the IDS/II database.

The dynamic database concept is used describe real database operations, including any specific record-occurrences and the day-to-day changes which need to be made.

This manual is organized in the following way:

Section II discusses IDS/II from the point of view of a static database; that is, it focuses on the organizational structure of the IDS/II database in terms of records, sets, and record-set constructs;

Sections III, IV, V, VI and VII, however, discuss IDS/II from the point of view of a dynamic database which actually changes and requires modifications.

Apart from the static record-set organization given in **Section II**, the remaining sections of this manual describe the dynamic (or real) database, by section, as follows:

1) **Section III** - Preliminary notions:

    Areas
    Data-base-key and Area-key
    Record location modes
    Secondary keys Indices
    Program/DBCS communication
    Currency Indicators
    Set membership class
    Set selection criteria
    Set ordering criteria
    No-duplicates control
    Validity checks on data fields

2) **Section IV** - The Subschema Model:

The Subschema concept

The Construction of a Subschema

3) **Section V** - Data Manipulation Retrieval Functions:

FIND, GET, ACCEPT, Database Condition (IF)

4) **Section VI** - Data Manipulation Update Functions:

STORE, MODIFY, ERASE, CONNECT, DISCONNECT

5) **Section VII** - Data Manipulation Control Functions:

READY, FINISH, USE

6) **Section VIII** - Complementary notions in the IDS/II model:

a) Concurrent access to the same database from several run-units

b) Communication between programs accessing the same database within the same run-unit

c) Accessing several databases from one run-unit

d) Run-time consistency checks between IDS/II components

The concepts in this manual are introduced a progressive linear manner. However, since most of the subject matter presented is closely interrelated, forward references are often required. The reader who is not familiar with IDS/II is advised to read the manual through, in order to get a general idea of its contents.

The main concern of this manual is IDS/II terminology and methodology. Therefore, it does not discuss DDL, SDDL and DMCL in part of the manual and DML in another. For example, notions such as set membership class or set ordering and selection criteria, which can hardly be expained without referring to the dynamic aspects of the model, are in fact defined in the Schema DDL. Consequently, it is a good idea to to consult the appendices in Volume 1 of the *IDS/II Reference Manual* when reading the *IDS/II User 's Guide*, to help clarify the elements of DDL, SDDL, DMCL and DML syntax.

1.4.2 Place of the Manual in a Training Program

As explained above, this manual has a linear rather than a didactic structure. In most cases, new concepts are dealt with in depth as soon as they are introduced. It is therefore advisable that the newcomer to IDS/II have a solid overview of the product before proceeding to a detailed study of this manual.

Once the user has acquired a good knowledge of IDS/II concepts in this manual, he should refer to Volumes 1 and 2 of the *IDS/II Reference Manual* to learn the DDL, SDDL, DMCL and DML syntax in which the concepts are expressed, the operating procedures of the required processors and of DBCS run-time routines. A third manual, the *IDS/II Database Administrator's Guide,* provides complementary information on database integrity and performance and on the operation of IDS/II utilities.

The user should then begin writing sample schemas and DML programs and running them against small databases. Once the basic techniques involved in storage, retrieval, modification or deletion of records are well mastered, the user can begin designing database applications. This requires a preliminary study of your current or planned structural model and its operating procedures, since the database is supposed to simulate those procedures. Then, in light of the requirements of your DML programs, many decisions must be made concerning such areas as the choice of record-types, set-types, access paths to records (direct or via sets), physical placement and loading factors. Use of simulation tools, such as the SIMULOAD command of DBUTILITY, and testing of DML programs against a small-scale experimental database are prerequisites to the operation of the final database.

1.5 ABBREVIATIONS

Abbreviations used in this manual are listed and defined below:

CRU	Current of Run Unit
DBA	Database Administrator
DBCS	Database Control System (Run-time routines)
DBUTILITY	Database Utility
DBREORG	Database Reorganization Utility
DDL	Data Description Language
SDDL	Subschema Data Description Language
DDPROC	DDL/SDDL/DMCL Processor
DMCL	Device/Media Control Language
DML	Data Manipulation Language
GAC	General Access Control
IDS, IDS/II	Integrated Data Store
TDS	Transaction Driven Subsystem
TPR	Transaction Processing Routine
UWA	User Working Area

2. Structural Model for the Organization of the Database

This section presents the two basic elements of the data structure:

- **Records**, which represent the entities of real data to be structured within the database;

- **sets**, which represent the relationships between these entities.

It describes, from a static point of view, the characteristics of each of these elements and the rules that govern their combinations.

2.1 RECORDS

2.1.1 The Concept of a Record

Basically, **records represent items or elements that exist in the business or organizational environment**: customers, suppliers, products, orders, invoices, departments, employees, universities, teachers, students, etc.

Records can represent **classes** of these items or elements rather than individual entities. For example, in an inventory control application of IDS/II, there is not one BOLT record for every bolt in the shop but one BOLT record for all the bolts. In other words, the quantity of bolts is an attribute of the record.

Records can also represent abstract entities which are not items elements: a color, an age, a date.

In certain circumstances, records may be created for the sole purpose of adjusting these real items to the database constraints of the record-set model; for example, **relator** records and **root** records, which might link the **order date** to a specific customer order for bolts. Examples of such records will be given in the following discussion.

In this manual, records are represented by rectangles as shown in Figure 2-1.

CUSTOMER-1
DEPARTEMENT-7
EMPLOYEE-725
ORDER-10
CUSTOMER-9

Figure 2-1. Records, shown as rectancles as throughout this manual

Each record is composed of zero, one, or several fields whose values serve to characterize and possibly uniquely identify the corresponding entity; an example would be, customer-number, customer-address, order-number, order-date, order-quantity.

2.1.2 The Concept of a Record-Type

Record is a general term, on the same level as **set**. If discuss a particular record, customer SMITH, for example, we will speak of the customer **record occurrence** "SMITH".

Record occurrences representing the same kind of entity are grouped together in a **record-type** . Each record-type is given a name in the Schema DDL. This means that all the customer records constitute the record-type "CUSTOMER" in IDS/II, and all the order records belong to the record-type "ORDERS".

Figure 2-2 shows this classification of records by type.

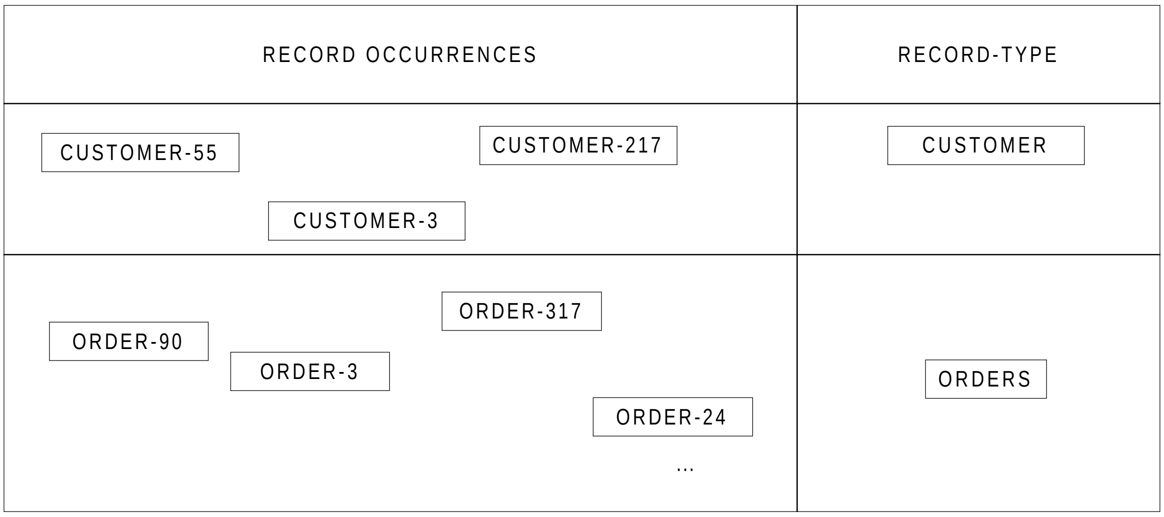


Figure 2-2. Record-type Versus Record Occurence

2.1.3 Record Description in the Schema

For each record-type, the Schema DDL specifies the structure common to all the records of this type in terms of field names and field characteristics. Fields can be classified in two categories, data items and data aggregates, as described in the corresponding subsections below.

2.1.3.1 Data Items (Elementary Fields)

Data items are the smallest units of named data. The Schema TYPE Clause specifies whether the data item is a **numeric data item** (decimal or binary) or a **string data item**. These different types of items are described below.

DECIMAL DATA ITEMS

To define the characteristics of decimal data items, the TYPE Clause has the following format:

```
          { SIGNED      UNPACKED  }
          {[UNSIGNED]   UNPACKED  }
TYPE IS   { SIGNED      PACKED    }    DECIMAL       m [ ± p ]
          {[UNSIGNED]   [PACKED]  }
          {[UNSIGNED]   PACKED-2  }
```

The meaning of "m" and "p" in the TYPE Clause format is:

m - the number of digits (m < 31).
p - the scale factor.

The m[**+** p] variable indicates the position of the (virtual) decimal point with respect to the rightmost digit. If p is not specified, the decimal point is immediately to the right of the rightmost digit. If +p is specified, the decimal point is *p* positions to the left; if -p is specified, it is *p* positions to the right. The information related to the scale factor does not appear in the data item itself but is recorded in a descriptor in the Schema.

Table 2-1 shows the correspondence between the DDL type and the DPS 7XXX machine format (as it appears in main memory as well as on disk).

Whatever the format, the following validity constraints apply when decimal data enters the database:

- The only legal digit values are 0, 1, 2 ...9.
- The only legal sign values are (in hexadecimal) A,C,E,F, for "+"; B,D, for "-". Depending upon the format, the "+" sign may be further constrained to only one value: F or C.

The decimal formats differ in the presence or absence of the sign and in the compaction of the digits.

The UNPACKED format is a display format (1 digit per byte). The "zone" parts of operands which are input to decimal operations are ignored. However operands that contain the result of a decimal operation have a "normalized" zone pattern of hexadecimal F.

The PACKED format is oriented towards storage compaction (2 digits per byte). If SIGNED is specified, the sign occupies one quartet. If UNSIGNED is specified, the sign is removed, which causes a gain of one byte when m is even. However the UNSIGNED PACKED format is not handled directly by the machine and necessitates a software conversion in computations and move operations. The UNSIGNED PACKED-2 format is a machine format including a positive sign, that is without optimization when m is even.

BINARY DATA ITEMS

To define the characteristics of binary data items, the TYPE Clause has the following format:

```
TYPE IS [SIGNED] BINARY {15}
                        {31}
```

This type applies only to integers.

As shown in Table 2-3, the number of bytes occupied is 2 or 4. This format is the most compact for a numeric item greater than 99.

STRING DATA ITEMS

To define the characteristics of a string data item, the TYPE Clause has the following format:

```
TYPE IS CHARACTER n [DEPENDING ON db-ident.]
```

If the DEPENDING ON Clause is not specified, this data item is a string of *n* EBCDIC characters.

If the DEPENDING ON Clause is specified, this data item is a variable-length string with a maximum of *n* EBCDIC characters and whose present number of characters is given by the value of the elementary field: db-ident.

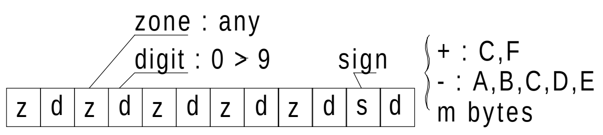
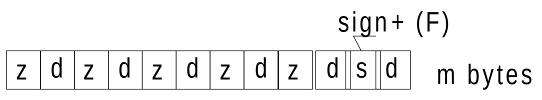
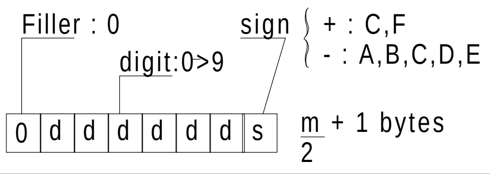
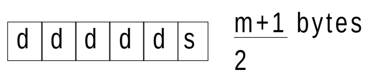
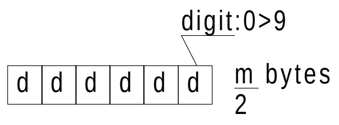
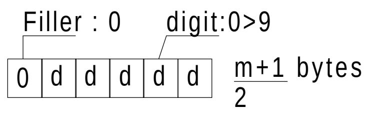
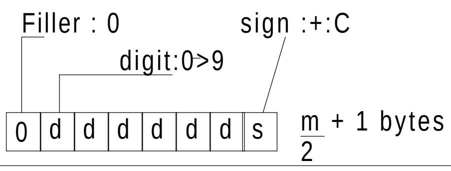
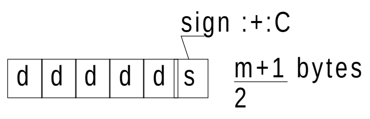
DDL TYPE	MACHINE FORMAT	REPRESENTATION
SIGNED UNPACKED DECIMAL m	unpacked decimal m	 zone : any digit : 0 > 9 sign + : C,F - : A,B,C,D,E m bytes z d z d z d z d z d s d
UNSIGNED UNPACKED DECIMAL m	absolutized unpacked decimal m	
SIGNED PACKED DECIMAL m	packed decimal m	m even (6)  Filler : 0 digit:0>9 sign + : C,F - : A,B,C,D,E 0 d d d d d d s m + 1 bytes / 2
		m odd (5) 
UNSIGNED PACKED DECIMAL m	none (COBOL) convention	m even (6) 
		m odd (5) 
UNSIGNED PACKED-2 DECIMAL m	positive packed decimal m	m even (6)  Filler : 0 sign :+:C digit:0>9 0 d d d d d d s m + 1 bytes / 2
		m odd (5) 

Figure 2-3. Correspondence Between DDL and Machine Data Types (1/2)

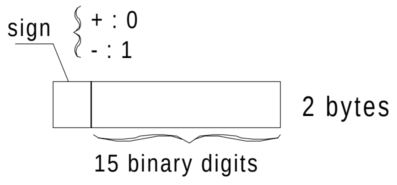
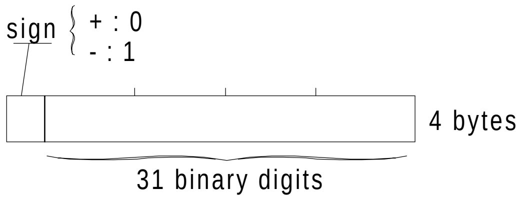
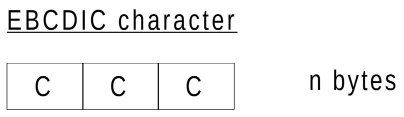
DDL TYPE	MACHINE FORMAT	REPRESENTATION
SIGNED BINARY 15	fixed binary 15	
SIGNED BINARY 31	fixed binary 31	
CHARACTERE n	EBCDIC character n	

Figure 2-3. Correspondence Between DDL and Machine Data Types (2/2)

2.1.3.2 Aggregates

These are named collections of data items. There are three kinds: vectors, repeating groups and non-repeating groups. The data items that are components of data aggregates are byte-aligned without synchronization on a word boundary.

VECTORS

A vector is a one-dimensional sequence of data items, all of which have identical characteristics. A vector is specified in the Schema by using three clauses:

- a TYPE Clause that specifies the data item characteristics,
- an OCCURS Clause that specifies the [fixed or maximal] number of items in the vector,
- and an optional DEPENDING ON Clause that specifies the present number of items in the vectors.

Examples:

```
ADDRESS TYPE IS CHARACTER 20   OCCURS 3 TIMES.

ADDRESS TYPE IS CHARACTER 20   OCCURS 9 TIMES
DEPENDING ON ADDRESS-NUMBER.
```

REPEATING GROUPS

A repeating group is a collection of data that occurs a fixed number of times. The collection may consist of data items, vectors, repeating groups or non-repeating groups. Repeating groups can thus be nested.

A repeating group is characterized in the schema by an OCCURS Clause that specifies the [fixed or maximal] number of occurrences of the group, by the absence of a TYPE clause, and by an optional DEPENDING ON clause that specifies the present number of occurrences of the group.

Grouping is indicated by level numbers (01 through 99), the lower level numbers corresponding to the more inclusive groups.

Example:

```
01        A   OCCURS 3.              (repeating group)
  02      B   TYPE CHAR 2.           (data item)
  02      C.                         (non repeating group)
    03    D   TYPE CHAR 7.           (data item)
    03    E   TYPE CHAR 30 OCCURS 2. (vector)
```

NON-REPEATING GROUPS

A non-repeating group is a collection of data that occurs only once. The collection may consist of data items, vectors, repeating groups or non-repeating groups. A non-repeating group is characterized in the schema by the absence of both TYPE and OCCURS clauses.

Example:

```
01        K.                         (non repeating group)
  02      L   TYPE DEC 4 OCCURS 2.   (vector)
  02      M.                         (non repeating group)
    03    N   TYPE BINARY 31.        (data item)
    03    P   TYPE CHAR 6.           (data item)
```

SUBSCRIPTING

When data items that are components of vectors or repeating groups are referenced in DDL clauses, subscripts must be used to identify them. Subscripts are cited left to right starting from the most inclusive group. In the examples above, references to data items would be written: ADDRESS (3), B(2), D(3), E(3,1), L(1).

Non-repeating groups do not introduce subscripting levels.

The combinations of TYPE and OCCURS Clauses that characterize the various fields are summarized below:

Field category	Type clause	Occurs clause
data item	YES	NO
vector	YES	YES
repeating group	NO	YES
non-repeating		
group	NO	NO

2.1.4 User Working Area Record Description

For each subschema record-type referenced by a COBOL program, a record description is automatically generated in the Working Storage or Linkage section, referred to as the User Working Area (UWA). This record area serves as a communication zone between the program and the DBCS. It is there that the program prepares the field contents before STORE or MODIFY statements and reads the field contents after a GET statement.

The structure of the UWA record is exactly the same as the structure of the subschema record, except that it is expressed in COBOL terms rather than DDL terms.

Figure 2-4, further on, shows the equivalence between DDL and COBOL data types.

Example DDL record:

```
RECORD R01

          ...
   02      A.
     03    B   TYPE SIGNED PACKED DECIMAL 5.
     03    C   OCCURS 2.
       04  D   TYPE CHARACTER 3.
       04  E   TYPE UNSIGNED UNPACKED DECIMAL 4,1.
     03    F   TYPE UNSIGNED PACKED   DEC 4,-1.
   02      G   TYPE BINARY 15 OCCURS 7.
```

Example COBOL UWA record:

```
01 R01.  ...
  02      A.
    03    B   PIC S9(5) USAGE COMP.
    03    C   OCCURS 2.
      04  D   PIC X(3).
       04  E          PIC 9(3)V9.
    03    F                PIC 9(4)P USAGE COMP.
  02      G   USAGE COMP-1   OCCURS 7.
```

Table 2-1. Equivalence Between DDL and COBOL Data Types

DDL TYPE			**COBOL TYPE**	
			PICTURE	**USAGE**
SIGNED			S 9(m)	
UNPACKED DECIMAL	m,p	m <p> p m = p m	S 9(m-p) V 9(p) S V 9(m) S P (p-m) 9(m)	
	m,-p		S 9(m) P(m)	
SIGNED PACKED m m,p m,-p	DECIMAL		same as SIGNED UNPACKED	COMP
UNSIGNED	m		9(m)	
UNPACKED DECIMAL	m,p	m <p> p m = p m	9(m-p) V 9(p)V 9(m)P (p-m) 9(m)	
	m,-p		9(m) P(p)	
UNSIGNED PACKED m m,p m,-p	DECIMAL		same as UNSIGNED UNPACKED	COMP
UNSIGNED PACKED-2 m m,p m,-p	DECIMAL		same as UNSIGNED UNPACKED	COMP-8
SIGNED BINARY 15				COMP-1
SIGNED BINARY 31				COMP-2
CHARACTER n			X(n)	

2.1.4.1 Variable Length Records

A variable length record is a record in which a DEPENDING ON db-ident clause is specified, and which obeys the following rules:

- at most 1 DEPENDING ON clause is authorized
- the variable data item, vector, or repeating group must be declared in last position of the record
- the db-ident must be:
    - a data item declared in the same RECORD Entry
    - a positive integer (decimal or binary)
    - an elementary field (not subscripted)

In the example below, **C** is a repeating group with a maximum of 20 entries. **B** represents a **DEPENDING ON** control field and **C** represents a controlled group. The number of current entries is indicated by the value of **B**.

```
RECORD R01

         ...
  02      A.
    03    B   TYPE SIGNED PACKED DECIMAL 5.
    03    C   OCCURS 20 DEPENDING ON B.
      04  D   TYPE CHARACTER 3.
      04  E   TYPE UNSIGNED UNPACKED DECIMAL 4,1.
```

In the example below, **F** represents an elementary field whose maximum length is 512; **B** represents the current length.

```
RECORD R01

         ...
  02      A.
    03    B   TYPE SIGNED BINARY 15.
    03    C   OCCURS 2.
      04  D   TYPE CHARACTER 3.
      04  E   TYPE UNSIGNED UNPACKED DECIMAL 4,1.
    03    F   TYPE CHARACTER 512 DEPENDING ON B.
```

2.2 SETS

2.2.1 The Concept of a Set

We can define a set as a **1-to-N** relationship between 1 **owner** record and N **member** records (including the case of a set with zero members).

Following are examples of possible sets represented in the database:

1) the orders placed by customer DUPONT. In this set, DUPONT is the owner of the relationship, his orders are the members of the relationship.

2) the employees of department ENGINEERING.

3) the courses attended by student JOHN.

4) the students attending the course HISTORY.

5) products of color BLUE.

6) persons born on the year 1967.

A set can be represented by a chain linking the owner and its members, as illustrated by Figure 2-5, below.

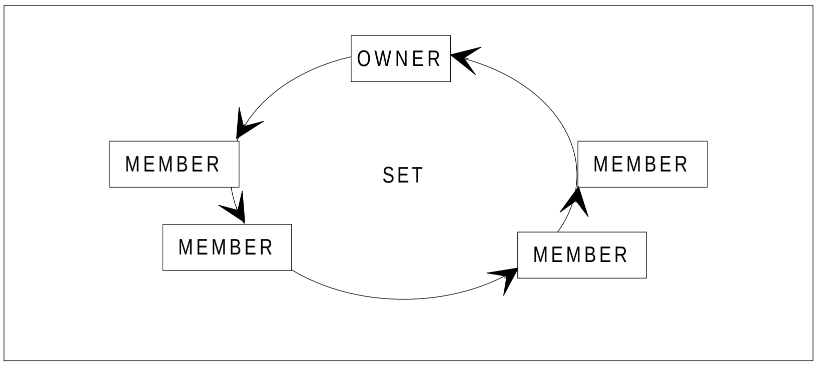


Figure 2-4. Set Representation

Figure 2-6, below, provides several examples of sets.

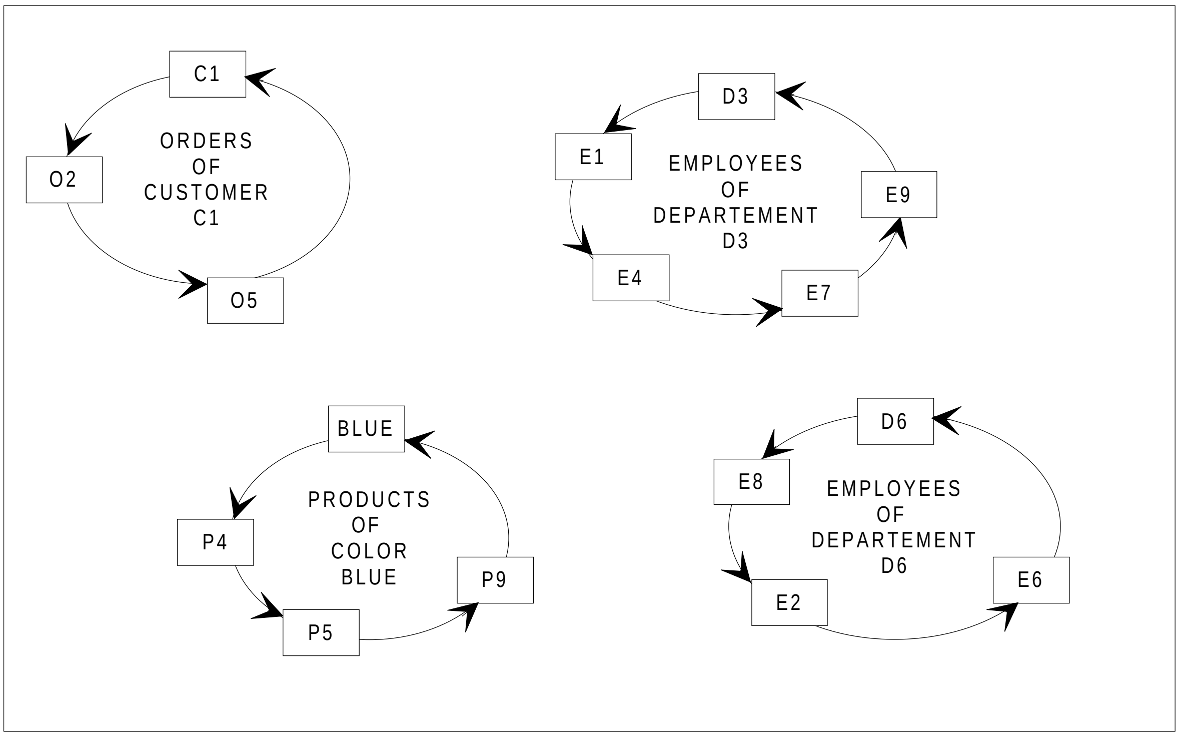


Figure 2-5. Examples of Sets

2.2.2 The Concept of Set-type

We have seen that the customer and order records are grouped into **CUSTOMER** and **ORDERS** record-types respectively. Since every customer is likely to place orders, it follows that all occurrences of the relationship "orders of customer X" constitute a "set-type", with the possible name **ORDERS-OF-CUSTOMER**.

More generally, **a set-type is a named relationship between one owner record-type and one or several member record-types**. It is represented in a symbolic way by an arrow instead of a chain, as indicated in Figure 2-7

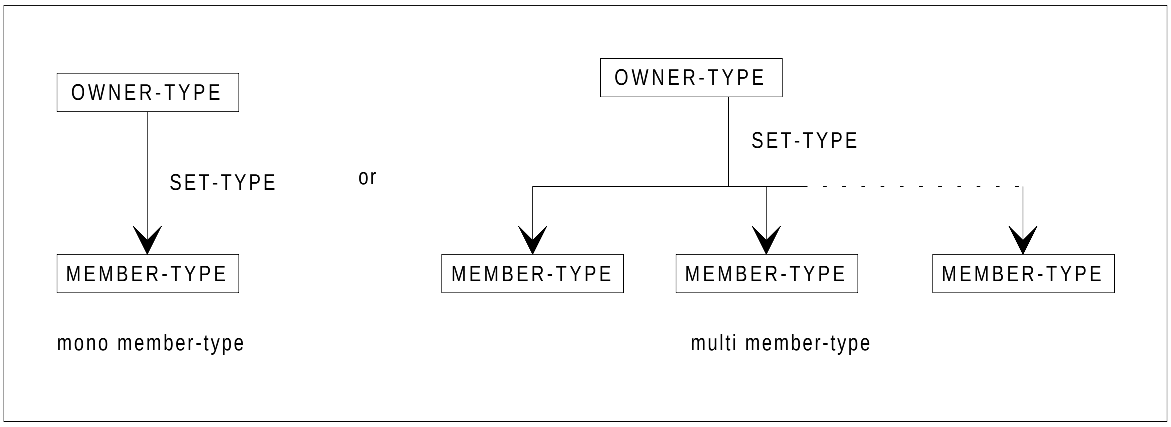


Figure 2-6. Set-type Representation

In this manual, we refer to diagrams representing set-types and record-types as **type diagrams** as opposed to **occurrence diagrams**. Type diagrams are used to show the organization of set-types, while occurrence diagrams show that of set occurrences.

Figure 2-8 provides an occurrence diagram and a type diagram showing a number of sets.

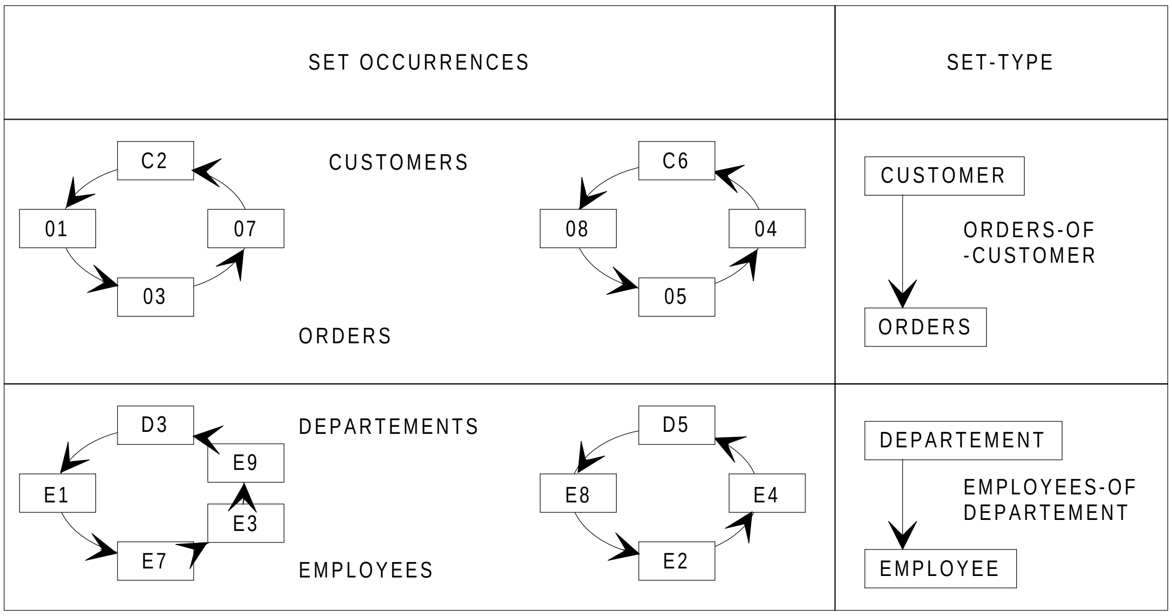


Figure 2-7. Occurence Diagram Versus Type Diagram

The **type diagram** is a concise way of representing a data structure. It proves particularly useful in the case of network structures, where an occurrence diagram would be too intricate. Set-type arrows represent a multitude of chains, each one linking an occurrence of the **owner** record-type to zero, one, or several occurrences of the **member** record-types. Because of its simple structure, the type diagram can be easily used to communicate database information to non-specialists or to those who are not working directly with IDS/II.

Two rules complement the definition of a set-type:

- Owner and member record-types must be different from one another. This forbids "recursive" sets, but we will see later that such a limitation may be circumvented by the introduction of **relator** records.

- The same record occurrence, whether of the owner or member record-type, cannot participate in two occurrences of the set-type. This ensures that set occurrences are disjointed, as shown in Figure 2-9.

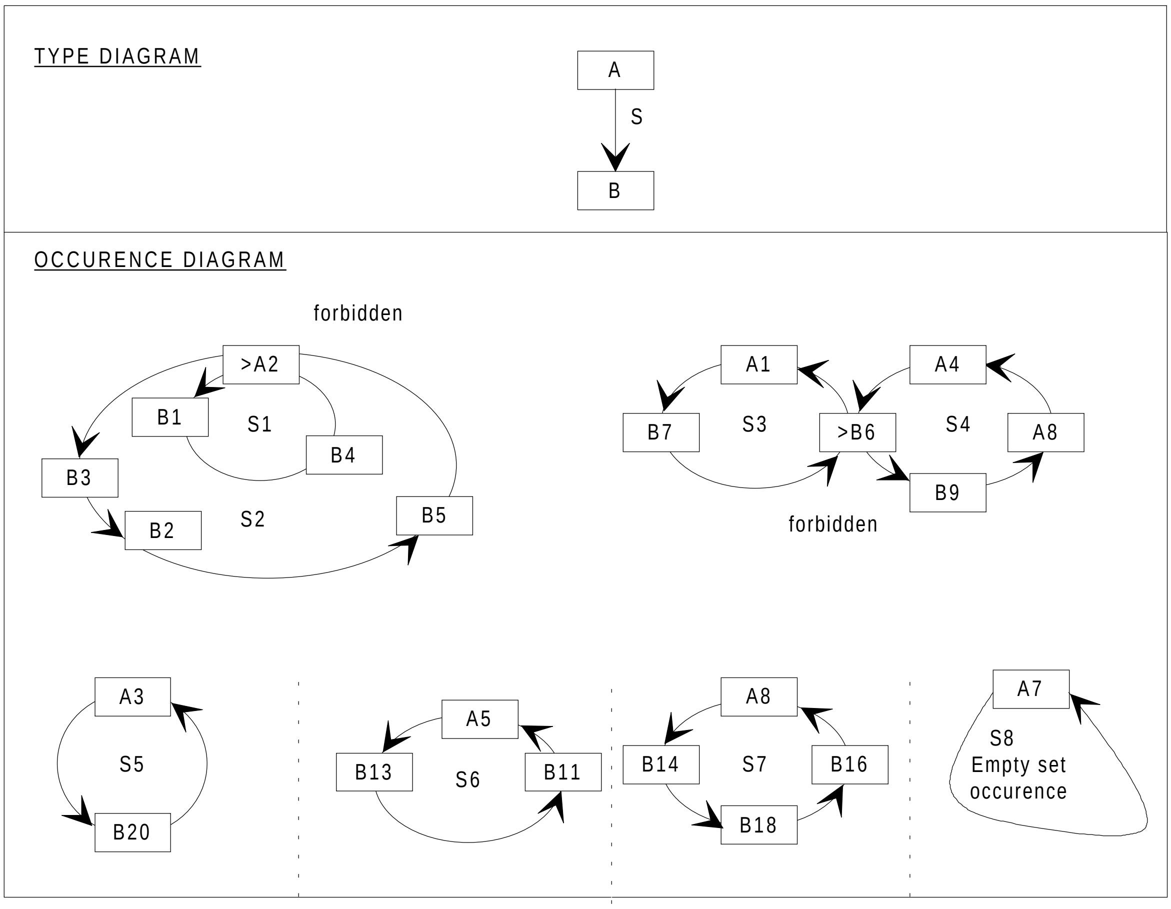


Figure 2-8. Separate Occurences of a Set-type are Disjointed

2.2.3 Example of Set Implementation

This subsection presents the concept of a **chain of pointers** in a set, explains the meaning of set order and the use of NEXT, PRIOR, FIRST, LAST records. These general concepts may be implemented by other methods (indexes, for example) in other database systems.

Set pointers are stored along with user data as a part of the storage records themselves, as shown in Figure 2-9

```
+-------------+      }
|   SET       |      }
| POINTERS    |      }
|-------------|      }      STORAGE RECORD
| USER DATA   |      }
|             |      }
+-------------+      }
```

Figure 2-9. Placement of Set Pointers with User Data

The sequential organization of the chain unambiguously defines the forward and backward directions used to search records. Thus, the terms NEXT, PRIOR, FIRST and LAST, used throughout the manual, are clear and unambiguous.

For each set of which it is the **owner**, a record contains the following pointers:

- A NEXT pointer, which points to the first member of the set (or to itself if the set occurrence is empty).

- A PRIOR pointer, which points to the last member of the set (or to itself if the set occurrence is empty).

For each set of which it is a **member**, a record contains the following pointers:

- A NEXT pointer, which points to the next member of the set (or to the owner if it is itself the last member of the set occurrence).

- A PRIOR pointer, which points to the prior member of the set (or to the owner if it is itself the first member of the set occurrence).

- An OWNER pointer, which points to the owner of the set.

The NEXT, PRIOR, and OWNER pointers of a MANUAL-OPTIONAL member point to the record itself if it is not currently connected. (See "Set Membership" in Section III).

Figure 2-10 shows the various pointers of a set occurrence.

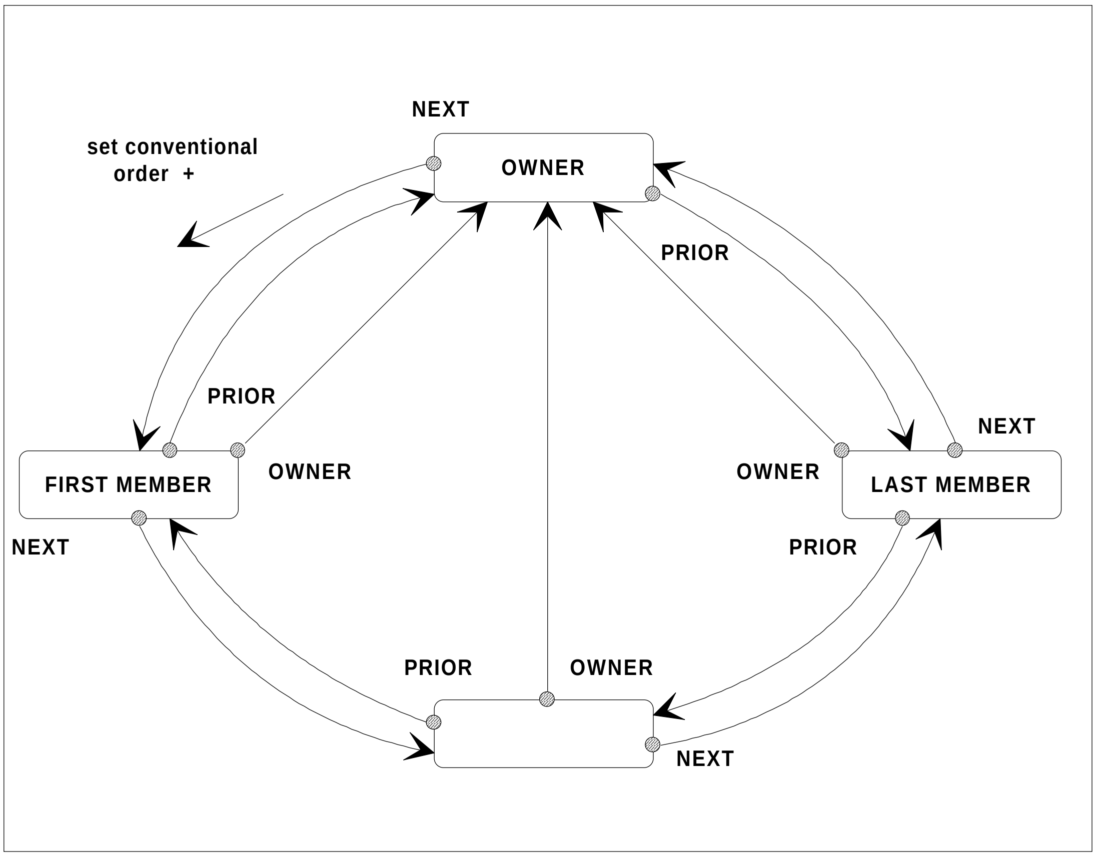


Figure 2-10. The Chain of Pointers in a Set

2.3 RECORD-SET CONSTRUCTS

2.3.1 Building Rules

A record-type can be of two forms:

- a member-type, having 0, 1 or m set-types
- the owner-type, having 0, 1 or n set-types.

Figure 2-12 illustrates these rules.

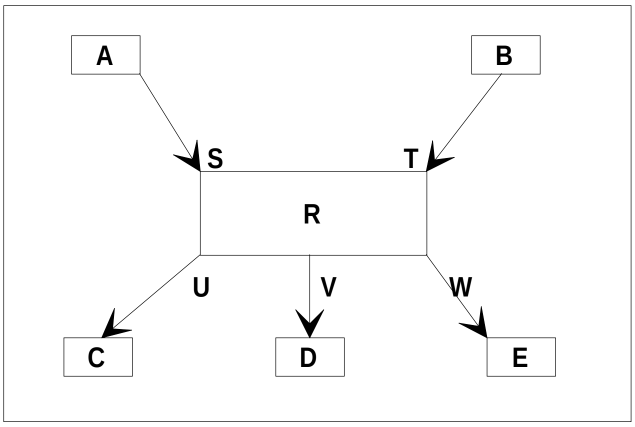


Figure 2-11. Participation of a Record-type in Several Set-types

NOTE: Record-types may exist which do not participate in any set-type, whereas set-types cannot be defined without reference to record-types.

2.3.2 Elementary Record-Set Constructs

Based on the rules we have discussed, some simple record-set constructs can be considered which serve as building-blocks for more elaborate models.

Figure 2-13 illustrates five elementary data structures, which are discussed in the subsections below:

- hierarchy structure
- cycle structure
- tree structure
- simple network structure
- complex network structure

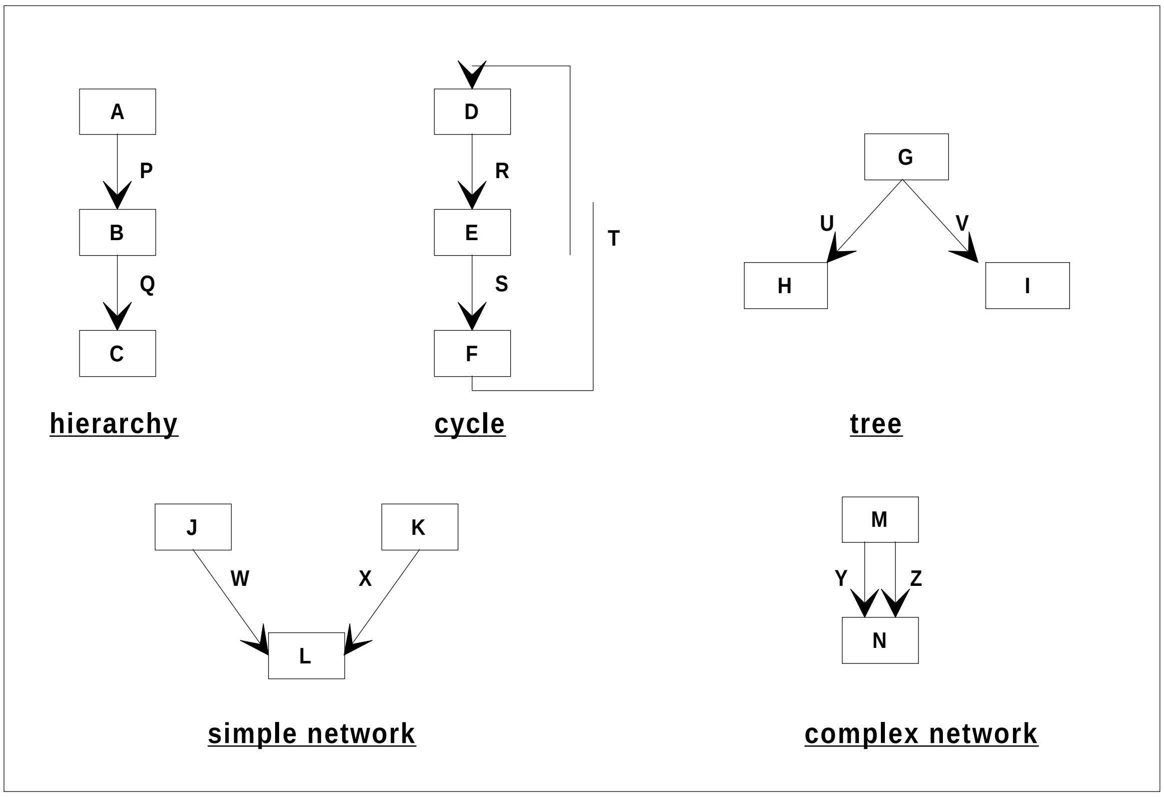


Figure 2-12. Elementary Record-Set Constructs

2.3.2.1 Hierarchy Structure

The corresponding **hierarchy** occurrence diagram is shown in Figure 2-14**.**

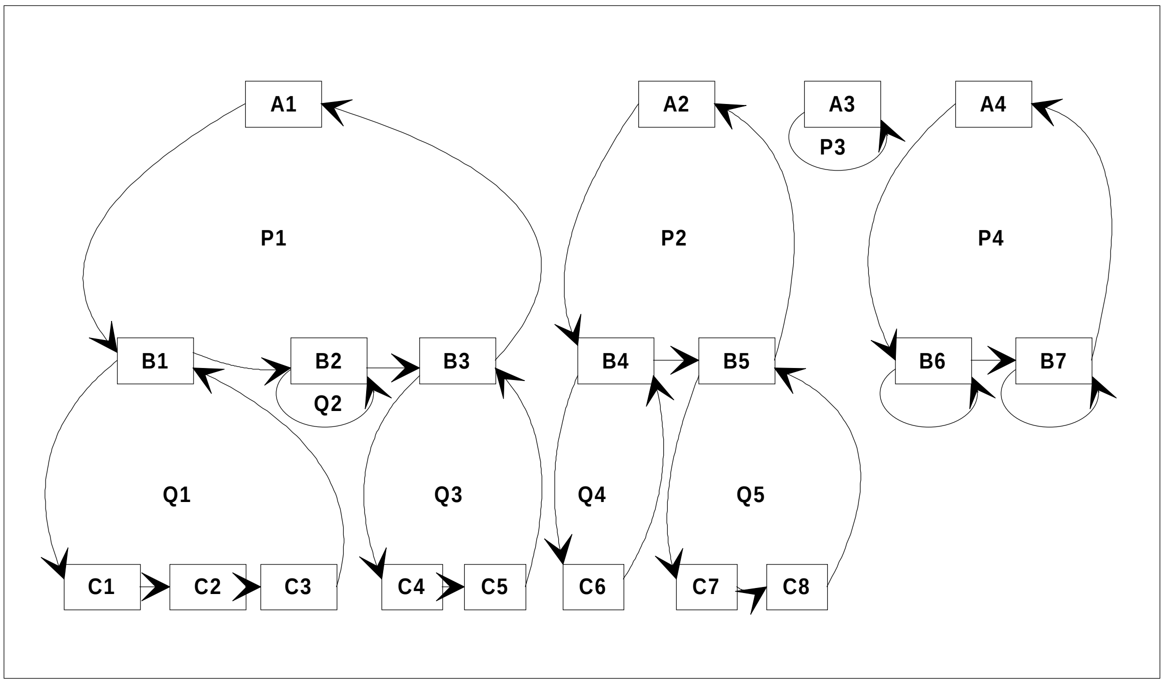
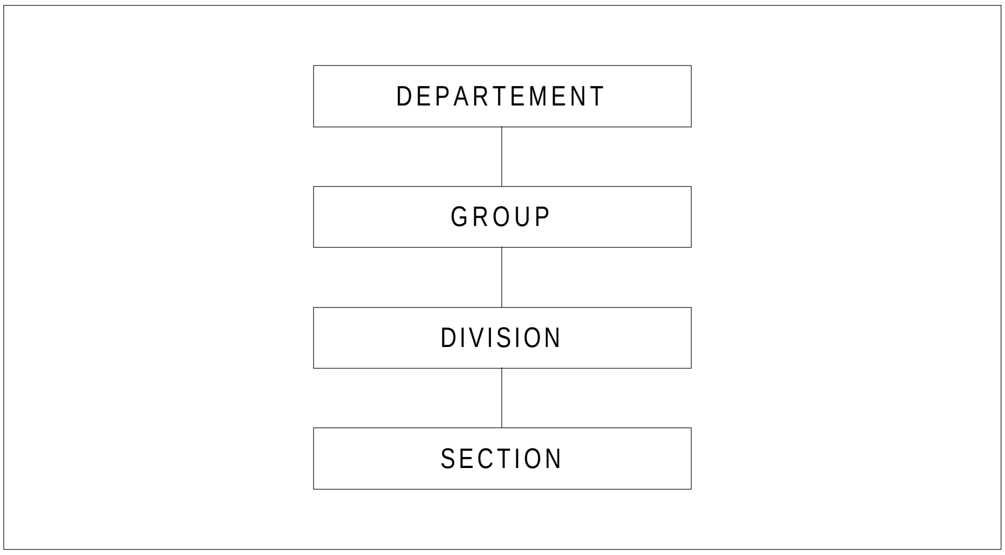


Figure 2-13. Hierarchy Occurence Diagram

The hierarchy structure is one of the most common. Organization-units can often be represented by type diagrams such as the one shown below:



2.3.2.2 Cycle Structure

The cycle structure is a series of sets which are defined such that the owner-type of each set-type is a member-type of the preceding set-type in the series.

A cycle structure requires that at least one of the member record-types be MANUAL (see "Set Membership" in Section **III**). MANUAL membership is represented by a broken arrow in the manuals of this series.

2.3.2.3 Tree Structure

Figure 2-15 shows a **tree** occurrence diagram.

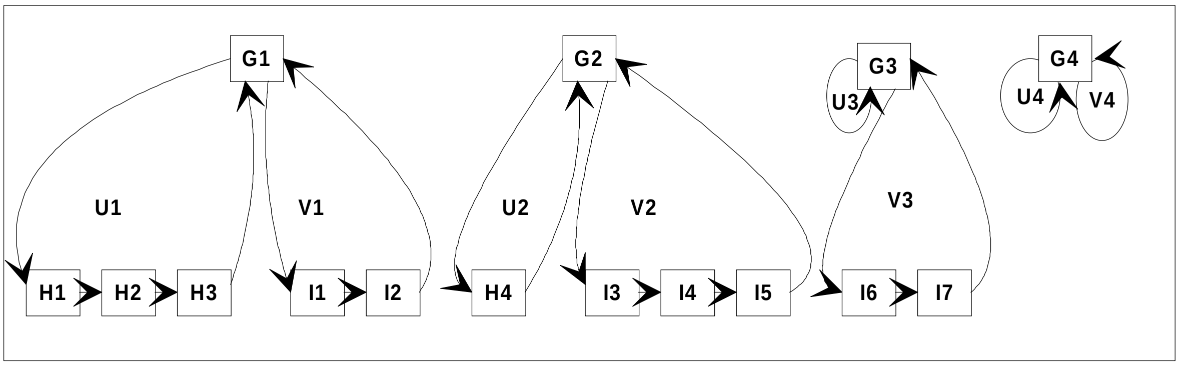


Figure 2-14. "Tree" Occurence Diagram

The tree structure is used when several lists of characteristics are attached to a record-type. For example, in Figure 2-15, record-type G may represent a taxpayer, set-type U the list of his sources of income, set-type V the list of the persons in his charge.

The tree structure is similar to a set-type with several member-types, as shown in Figure 2-16.

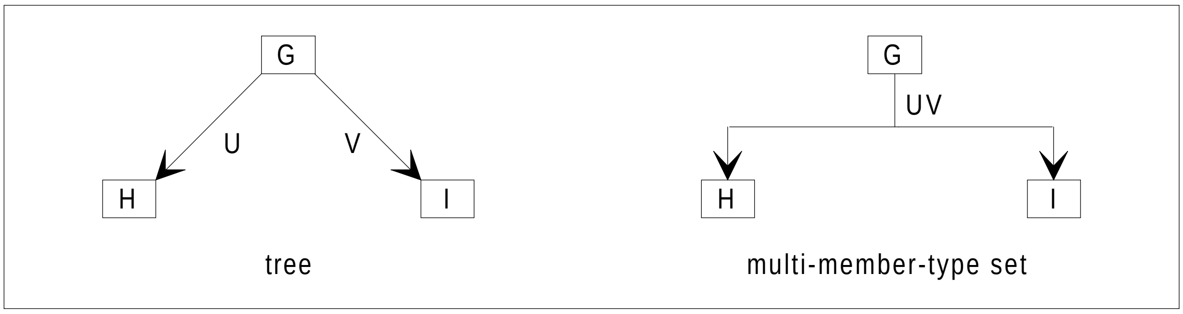


Figure 2-15. Tree Versus Multi-member-type Set

This similarity is due to the fact that, in the case of a multi-member-type set, different ordering and set selection criteria may be defined for each member-type and thus, DML retrieval statements searching the set may be restricted to only one member-type.

A Database Administrator might choose the multi-member-type set to make the best use of storage space, because this saves the creation of some pointers in the owner record-type. This is particularly true when the members of types H and I are few or when there are no members of types H and I at all.

2.3.2.4 Simple Network Structure

The term "simple" means only that the related occurrence diagram is fairly easy to understand. Figure 2-17 illustrates this by presenting the network in the form of a matrix where the L records are placed at the intersections of the W and X.

NOTE: The representation in Figure 2-17 is simplified, because the lines indicating the sets should be drawn diagonally between the L records in a row or column to demonstrate the ordering of these records.

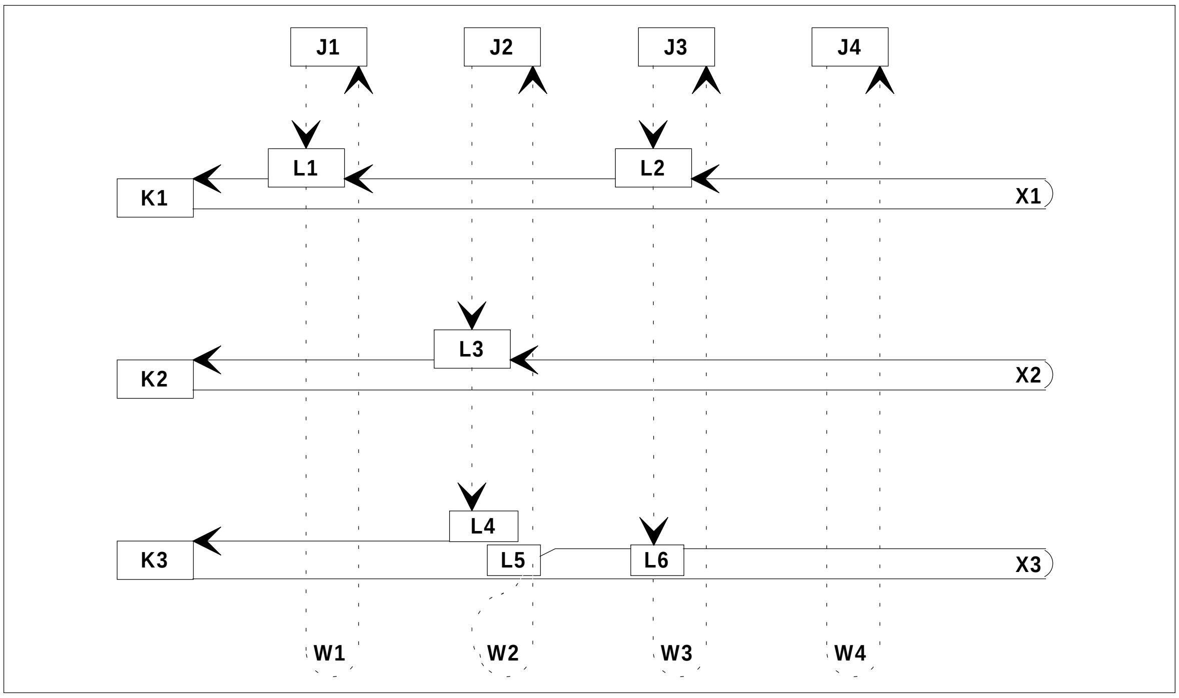


Figure 2-16. Simple Network Occurence Diagram

The simple network structure is common. We have already seen the example of an ORDERS record which is related to a CUSTOMER record and a PART record at the same time, as shown in Figure 2-18, below.

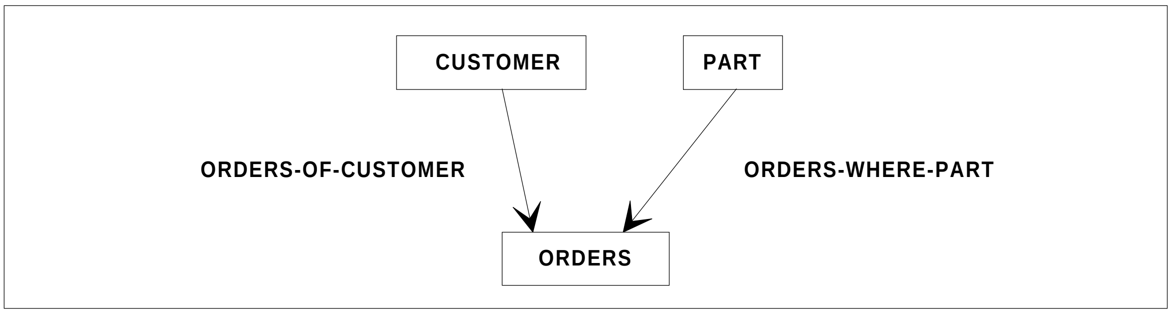


Figure 2-17. Costumer-Part Network Diagram

In many cases, **the relationship between two record-types is of the type "m to n"** rather than "l to n", as demonstrated in Figure 2-19 below, where records J and K are shown as related to one another. As the record-set model does not allow for this relationship, it is necessary to introduce **relator records** to transform an "m to n" relationship into two "l to n" relationships. The result is a simple network, as shown in Figure 2-19.

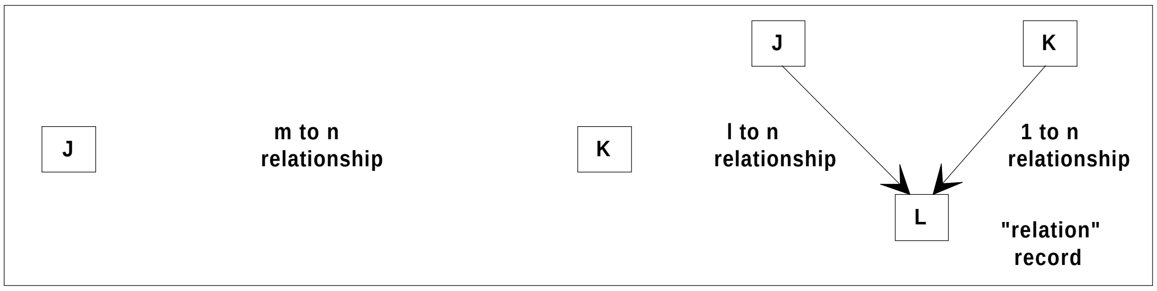


Figure 2-18. Simple Network Representing an "m to n" Relationships

Let us take the example of the relationship between the students and the courses of a university. Figure 2-20 shows which courses are attended by which students.

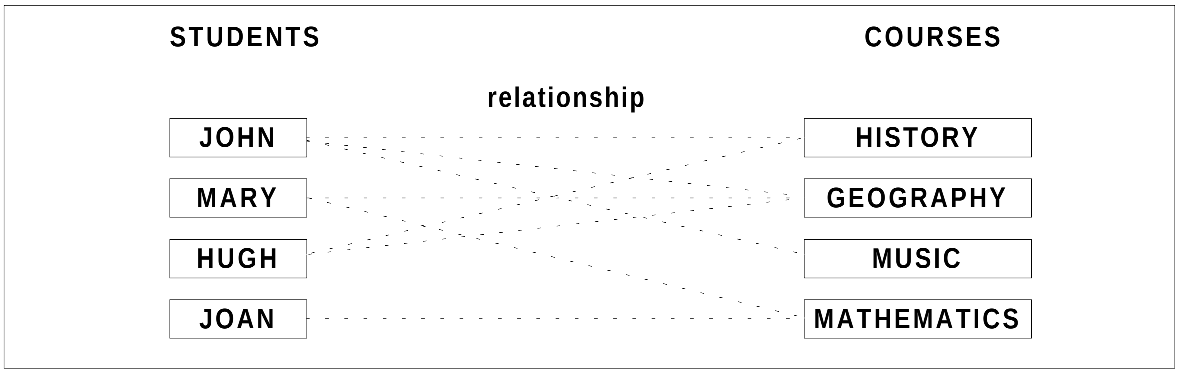


Figure 2-19. Example of "m to n" Relationships

Based on Figure 2-20, if we consider the STUDENT record as the owner of the relationship **courses attended by a given student**, it is clear that the COURSE record "HISTORY" would belong both to the set occurrence whose owner is JOHN and to the set occurrence whose owner is HUGH. This is against the rule which states that occurrences of a set-type are disjointed.

The same would be true if we considered the COURSE record as owner of a relationship **students attending a given course**.

The introduction of a **relator** record solves the problem. This record is made the member of the "l to n" relationships ATTENDS and ATTENDED-BY as shown by the type diagram in Figure 2-21.

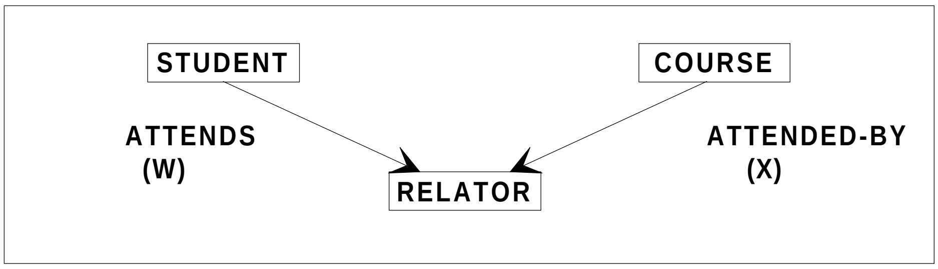


Figure 2-20. Relator

The simple network obtained is shown in the occurrence diagram of Figure 2-22, below.

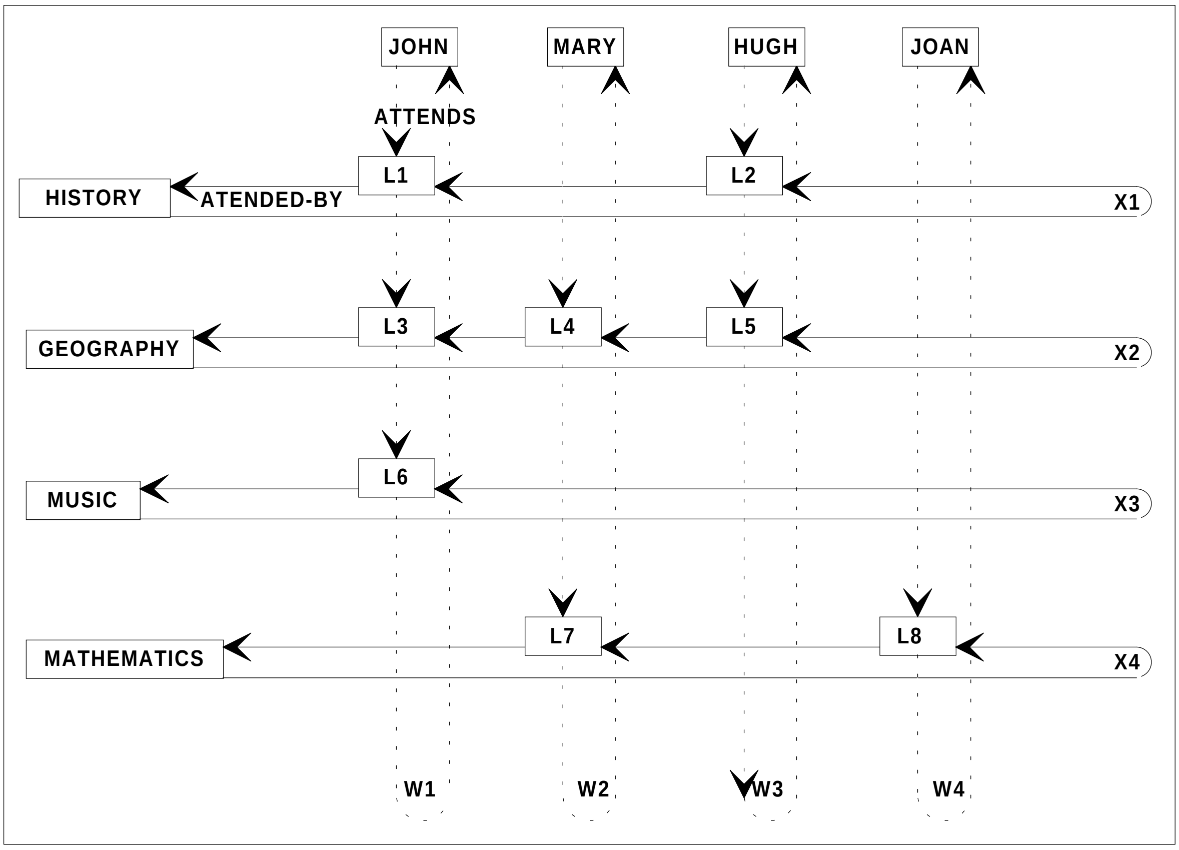


Figure 2-21. Simple Netword Between Students and Courses

Within the structure shown in Figure 2-22, if we want to find out which courses MARY attends, it is necessary to search the ATTENDS set occurrence W2 and, for each **relator** record found (L4, L7), to retrieve the owner of the corresponding ATTENDED-BY set occurrence (GEOGRAPHY for L4 and MATHEMATICS for L7).

A **relator** record may contain no user data if it serves only a structural purpose, but this is not the general rule. For instance, in the case of students and courses, the information concerning the results of examinations may very well be stored in the **relator** records.

2.3.2.5 Complex Network Structure

The term "complex" means only that the interpretation of the complex network occurrence diagram, shown in Figure 2-23, is not as straightforward as in the simple network case.

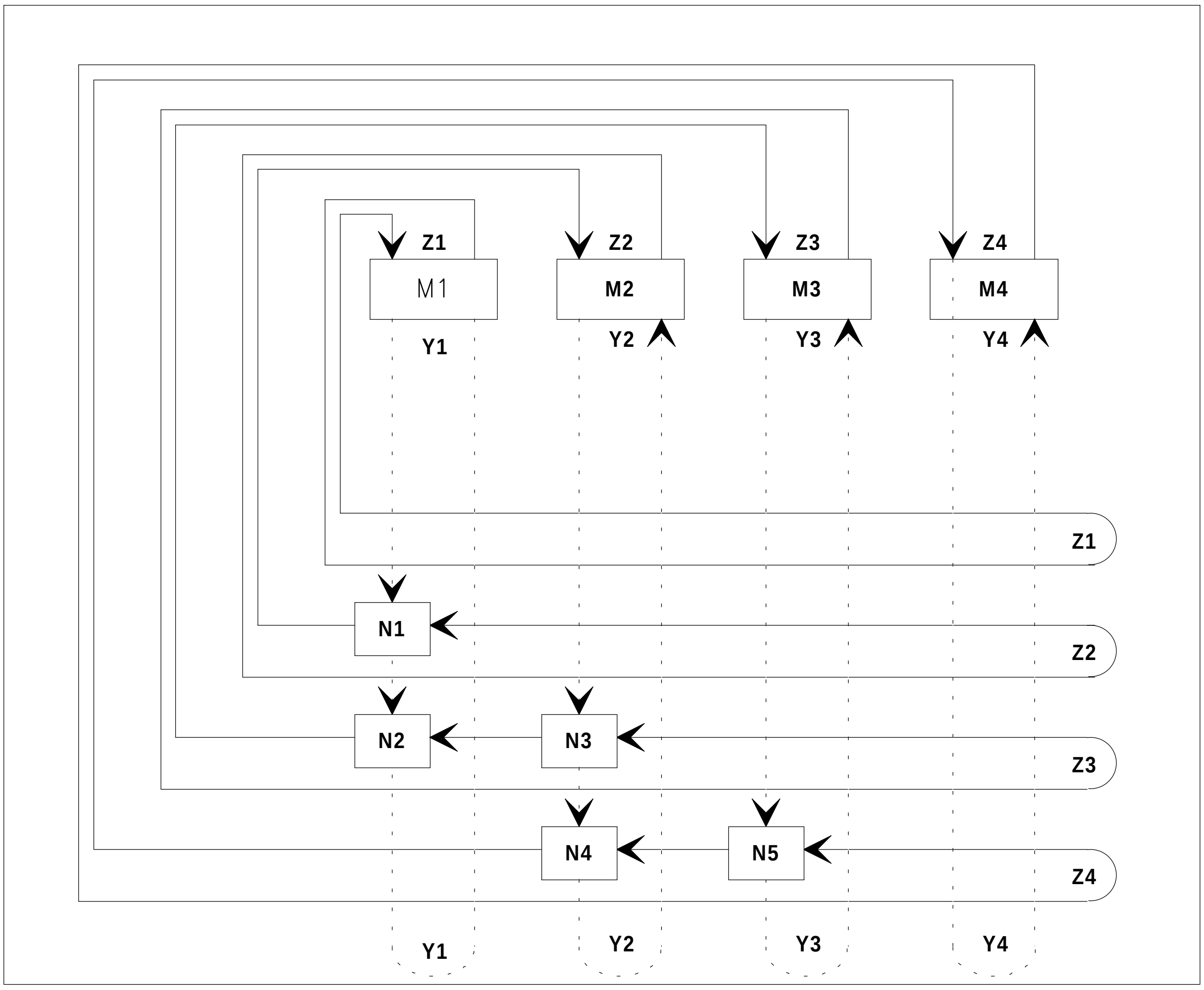


Figure 2-22. Complex Network Occurence Diagram

The complex network structure is often chosen because it can accomodate relationships between entities of the same nature which cannot be represented directly with the current record-set model. In other words, the complex network structure can accomodate relationships between occurrences of the same record-type. Contrary to the rules concerning the record-set relationship stated earlier, these relationships require that the same record-type be both owner-type and member-type of the same set-type. Figure 2-24 shows this construct, which is forbidden in IDS/II.

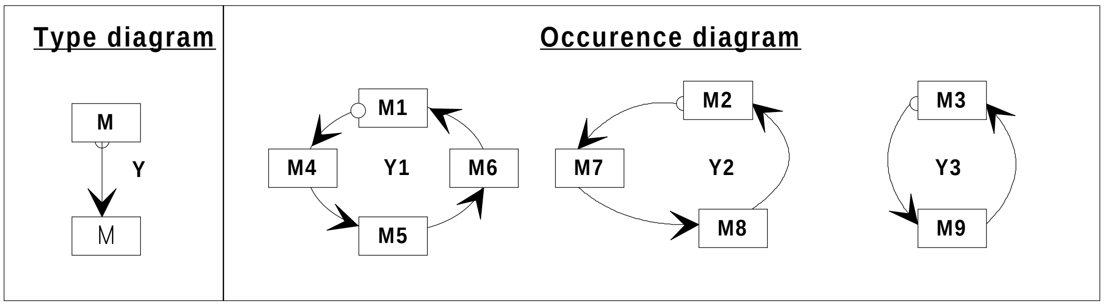


Figure 2-23. Forbidden Construct: Same Record-type as Owner and Member

Moreover, such relationships are often recursive. If the organization-units are considered as entities of the same type, there will exist a hierarchical relationship between the occurrences of this record-type. Except for those units at the top and the bottom of the hierarchy, each unit appears to be subordinate to another unit and superior to several other units. Figure 2-25 illustrates this case of "recursive" set.

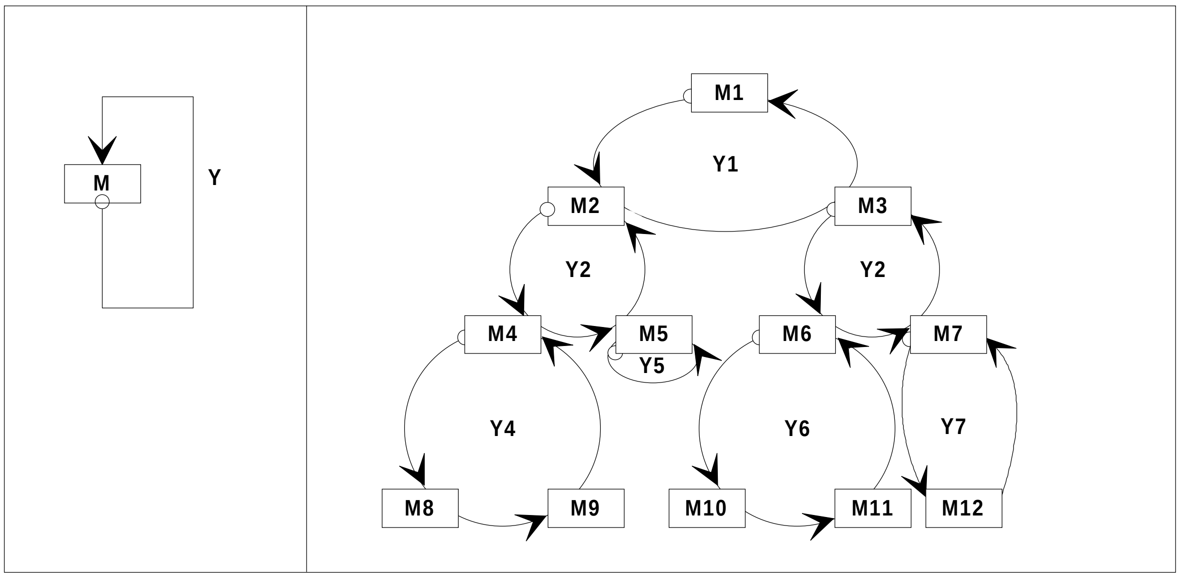


Figure 2-24. Forbidden Construct: Recursive Set

This construct also contradicts the rules for the record-set model already stated by requiring that the same record occurrence participate as member in one set occurrence and owner in a different set occurrence; in other words, these set occurrences would no longer be disjointed.

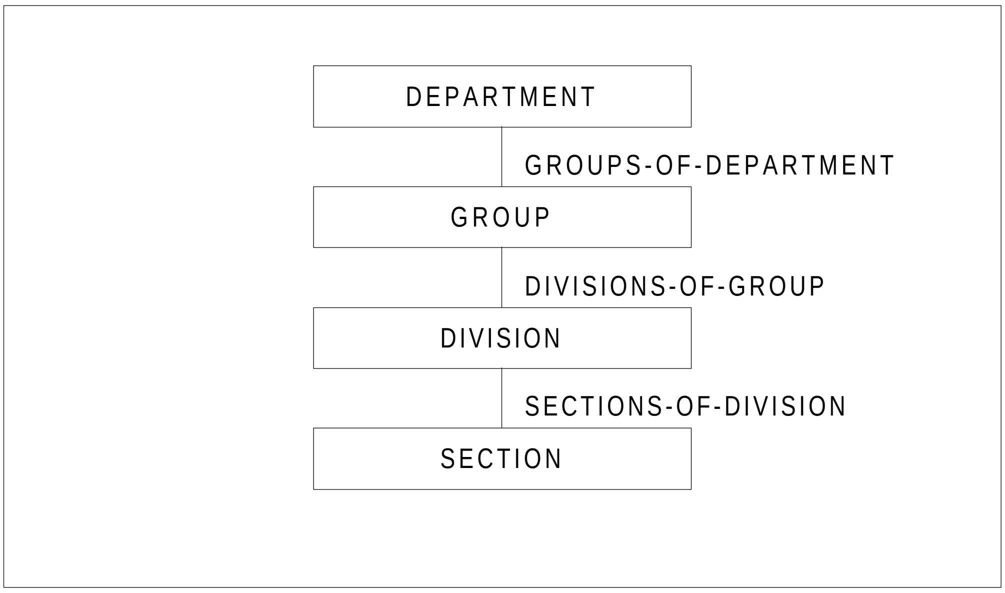
Another example of a recursive hierarchical relationship between entities of the same type is a bill-of-materials, which lists the elements required for the manufacturing of products.

In order to handle such situations, at least two methods can be used, which are explained on the following pages.

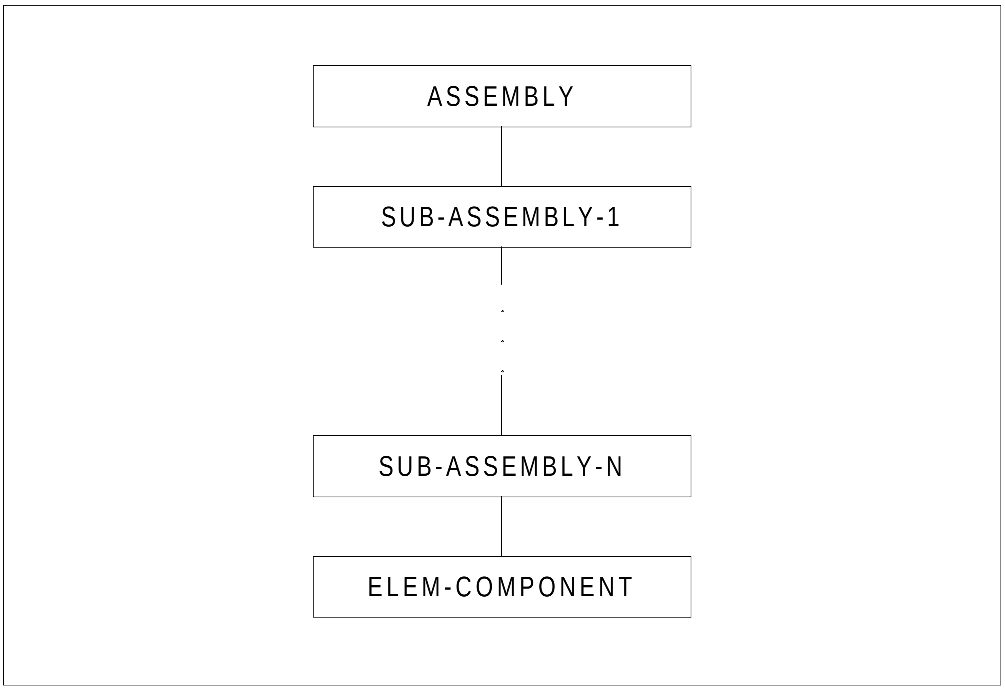
Method 1:

You create as many record-types as there are levels in the hierarchy.

The organization-units of the previous example could be classified according to the following diagram:



Similarly, the elements of a bill-of-material can be classified in assemblies, sub-assemblies of level 1,... sub-assemblies of level *n*, and elementary components. This structure is illustrated in the hierarchical structure shown in the diagram below.



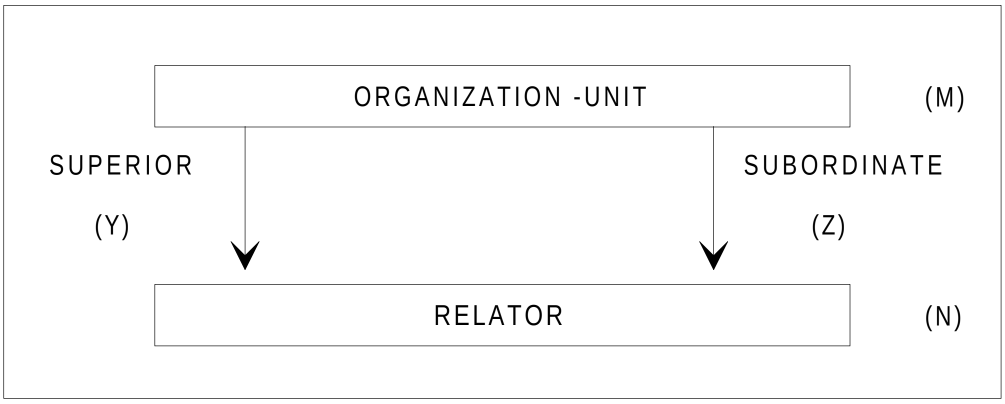
The disadvantages of this method are:

- A multiplicity of record-types and set-types which are required to represent the same kind of entity and relationship.
- A lack of flexibility if situations arise within the data relationships which present slight exceptions to a strictly hierarchical structure. This occurs in the structural model of the organizational structure if, for example, a section is tied directly to a group without an intermediate division. This can also occur more often in the bill-of-materials if a component is made up of sub-components selected at different levels of the hierarchy.

Method 2:

You design a complex network using only one record-type to represent entities of the same nature and one **relator** record-type to establish indirect relationships between these entities.

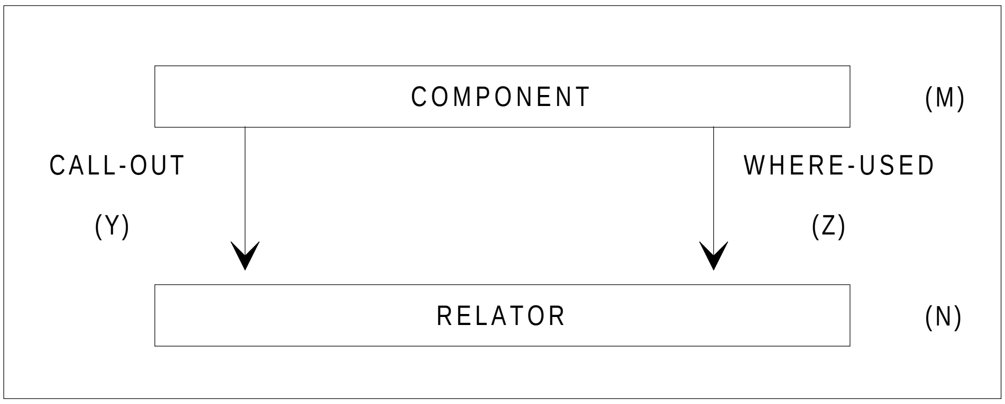
The organizational structure can be described by the diagram below:



To retrieve the organization-units that are subordinate to a given organization-unit (M1), a program has to search the SUPERIOR set of which M1 is the owner and, for each relator record found, find the owner of the corresponding SUBORDINATE set.

NOTE: In this example, a SUBORDINATE set contains at most one relator record if an organization-unit is tied to only one superior unit.

The bill-of-materials can be represented in a similar way:



For a given component in this model, the sets CALL-OUT and WHERE-USED indicate respectively its sub-components and the components of which it is itself a sub-component. The RELATOR record between a component and a sub-component usually contains the quantity of such sub-components which are necessary to build the component.

The occurrence diagram of Figure 2-26 illustrates the bill-of-materials network. It is presented in a way that shows the underlying hierarchical structure (assemblies, sub-assemblies, elementary components) but also the case of component M2 which is built from constituent M4 at the sub-assembly level and from constituent M7 at the elementary component level.

NOTE: At the ends of the hierarchy, the WHERE-USED set occurrences of the assemblies and the CALL-OUT set occurrences of the elementary components are empty.

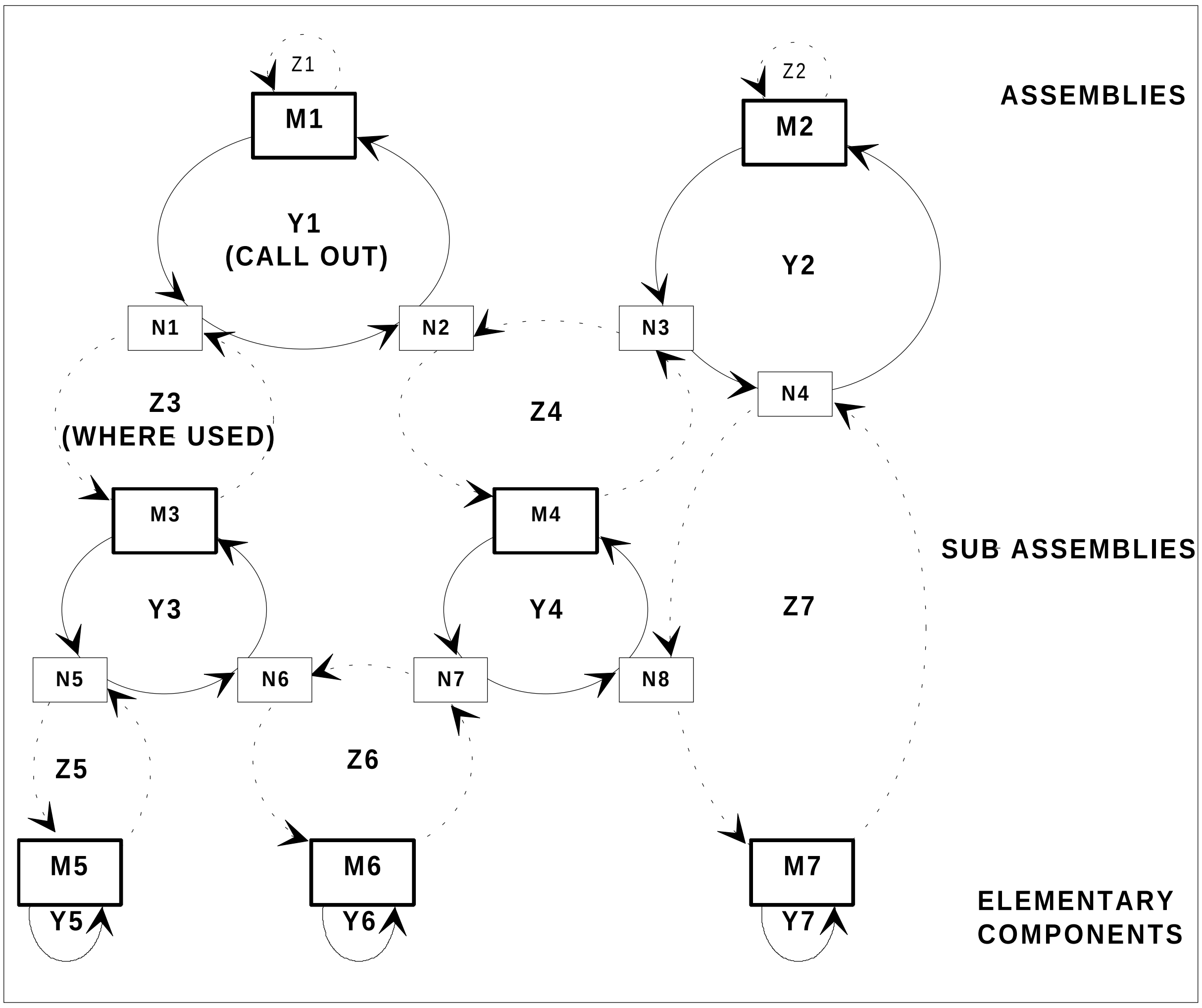


Figure 2-25. Bill-of-Material Network Occurence Diagram

The complex network structure is flexible because modifications of intra-record-type relationships can be handled at the occurrence level rather than at the type level; in other words, without afffecting the schema definition.

2.3.3 Example of a Record-Set Model

Figure 2-27 shows a record-set structure that contains the elementary record-set constructs discussed previously.

Despite its simplicity, this structure is general enough to be applicable to many enterprises.

At the center of the diagram is the bill-of-materials which represents the final products sold to customers, the elementary components bought from suppliers, and the sub-assemblies that correspond to intermediate steps in the manufacturing process.

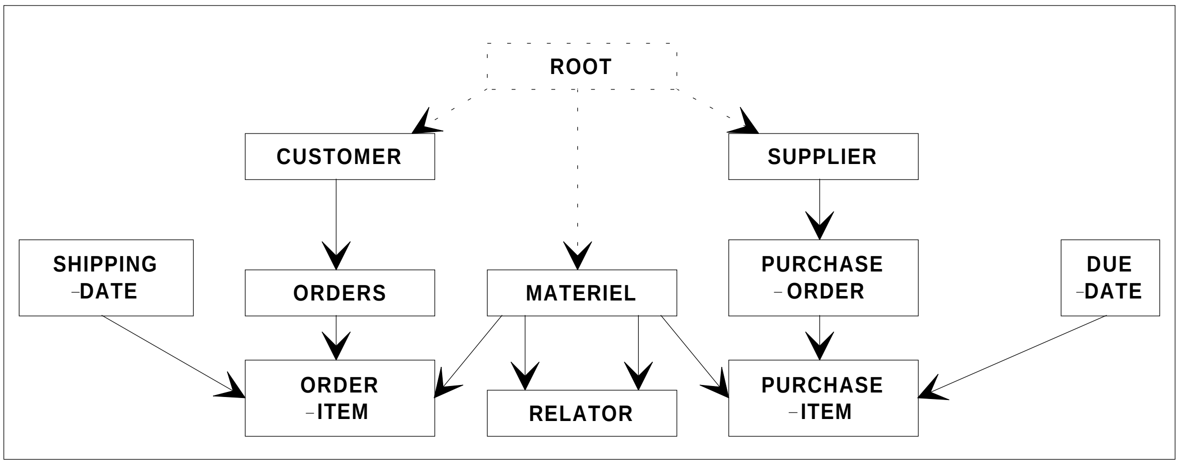


Figure 2-26. Example of Record-Set Model

On the left are represented the customers of the enterprise, the orders that they have placed, the detail items of these orders and the shipping date of the products ordered.

On the right are represented the suppliers of the enterprise, the orders that the enterprise has placed, the detail items of these orders and the due date of the components ordered.

At the top of the diagram, an optional artificial ROOT record groups all the customers, materials, and suppliers in sorted sets. In general, these sets have only one occurrence and are intended for global sorted reports.

2.3.4 Sets Versus Fields

To represent real situations, no single record-set model exists. The same information may reside in the form of fields in one model and in the form of sets in another model.

The choice between models depends on the following criteria:

- the need to make certain relationships explicit and use them as access paths.
- the need to solve problems which are caused by the constraints of the record-set data structure.

2.3.4.1 Example 1

The equivalence between fields and sets has already been illustrated in Section 1 by the example of an ORDERS record whose relationships to a CUSTOMER record and to a PART record are either:

a) implicity represented by a CUSTOMER-KEY fields and a PART-KEY in the ORDERS record or,

b) explicitly implemented by two sets ORDERS-OF-CUSTOMER and ORDERS-WHERE-PART.

See Figure 2-28, below.

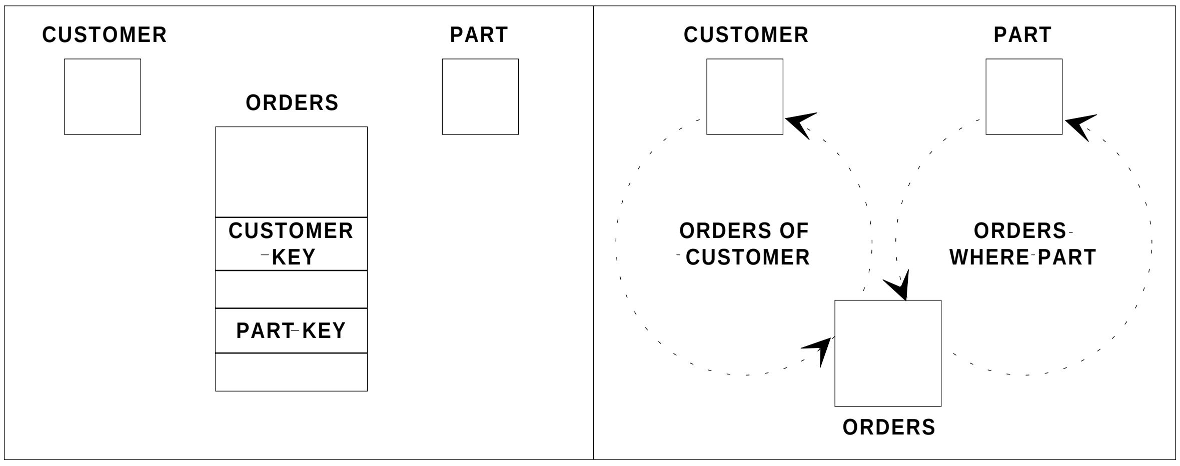


Figure 2-27. Field Versus Set Implementation

In the first case, given an ORDERS record, it is possible to directly retrieve the corresponding CUSTOMER record or PART record, since these records are likely to be directly accessible by their keys. But with a CUSTOMER record, the retrieval of all its associated ORDERS records will certainly involve a sequential search of the database.

In the second case, given an ORDERS record, the corresponding CUSTOMER record is retrieved immediately via the OWNER pointer of the ORDERS-OF-CUSTOMER set. As for the ORDERS records associated with a given CUSTOMER record, these can be easily retrieved by a search of the ORDERS-OF-CUSTOMER set occurrence, using the NEXT pointers.

2.3.4.2 Example 2

The fields of a record may represent attributes that are "specific" to the record. For instance a PERSON record may contain such information as birth-year, sex, marital status, nationality, etc.

If there is a need to know all the persons that were born in a given year or all the persons of a given nationality, the corresponding fields may be converted into sets that relate the PERSON record to a BIRTH-YEAR record and a NATIONALITY record. Figure 2-29 illustrates the transformation.

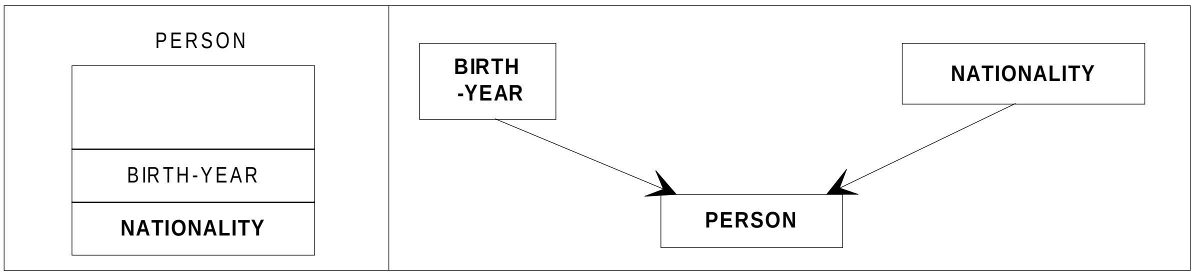


Figure 2-28. "Specific" or "Common" Attributes

Given a PERSON record in this model, the information concerning his birth-year or nationality is no longer found in the record itself but in the owner records of the corresponding sets.

To summarize, the **field model** attempts to characterize a record independently of all other records of its type. The **set model**, on the other hand, identifies collections of records that share the same values for some attributes.

2.3.4.3 Example 3

In the current release of IDS/II, records of the same record-type are fixed length. When the number of data items of a vector is variable, it may be necessary to convert these fields into records linked to their originating record by a set.

Figure 2-30 shows the example of a CUSTOMER record which is likely to contain more than one ADDRESS field. This field is transformed into an ADDRESS record which is made a member of a CUSTOMER-ADDRESSES set.

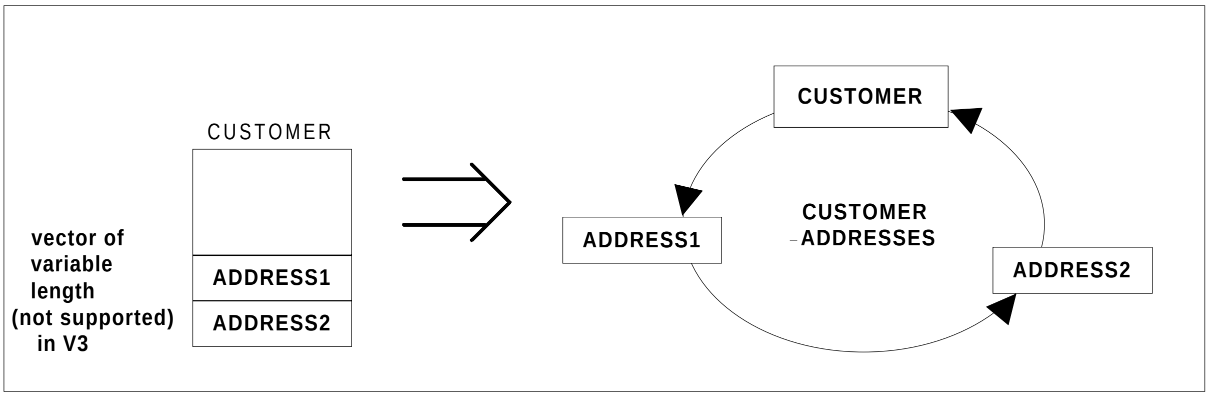


Figure 2-29. Set as Implementation of a Vector of Variable Length

NOTE: An intermediate solution consists of leaving the first address in the CUSTOMER record and using the set only for additional addresses.

However, since the majority of customers have only one address, the additional address set occurrences are often empty. In order to save the space occupied by unused pointers, it is possible to define two CUSTOMER record-types, one for a customer with one address, and one for a customer with several addresses. Only the second record-type may be owner of a set of addresses.

In this example, the record where the field-to-set conversion occurs becomes the owner of a new set. In the previous examples, it was made a member of a new set.

2.3.4.4 Example 4

Sets are defined to provide a link between records, not between fields. When the participation of a record in a set is due to only one of its fields, this does not cause any problem since there is a one-to-one correspondence between the field and the record. But when such a field happens to be repeated within the record, it is necessary to transform these fields into records.

Let us take again the example of an ORDERS record which is related both to a CUSTOMER record and a PART record, as illustrated by Figure 2-31.

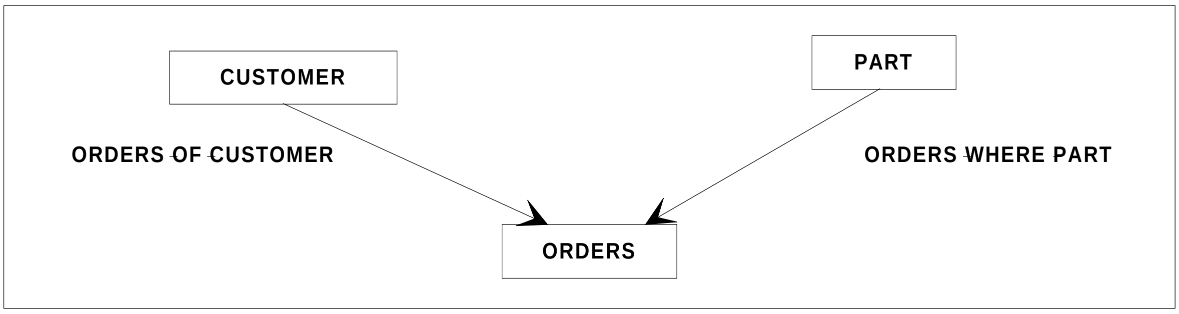


Figure 2-30. One ORDERS Record Corresponds to One PART Record

This diagram assumes that the ORDERS record contains a reference to only one PART. In a real situation, this is usually not the case; an order may contain several items, each referencing a different part. The present structure cannot be used any longer since the same ORDERS record would participate in several occurrences of the set ORDERS-WHERE-PART.

The structure shown in Figure 2-32 must be adopted to adhere to the record-set rules.

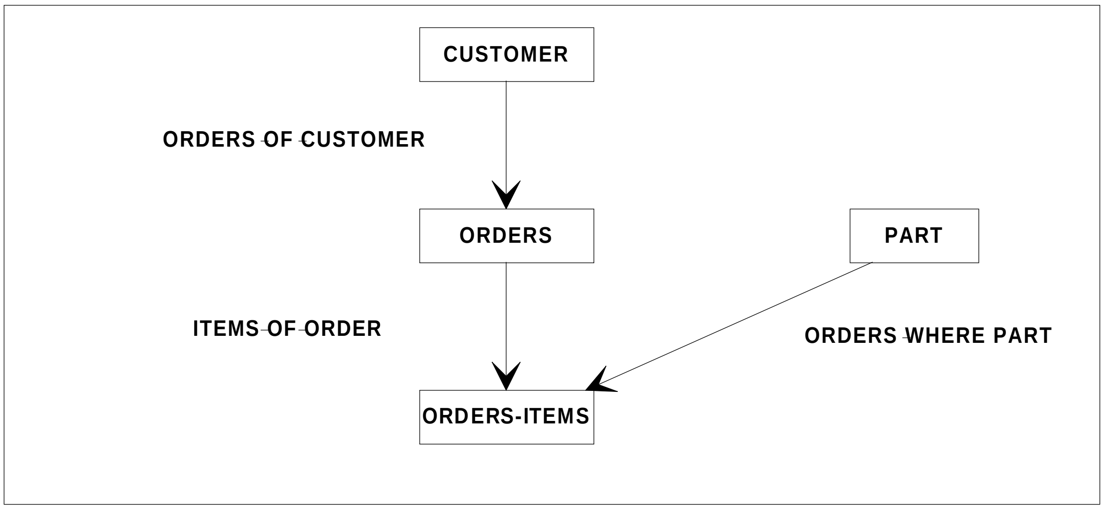


Figure 2-31. One ORDERS Record Corresponds to Several PART Records

In this diagram, each reference to a part becomes a separate record. This structure also solves the problem of having a variable number of items per order.

2.3.4.5 Conclusion

To summarize, let us list some advantages and drawbacks of sets with respect to fields.

- sets provide explicit relationships facilitating the database design.
- Advantages:
  - sets handle relationships by appropriate DML commands.
  - sets provide faster access path for retrieval.
  - sets can incorporate clustering.
- Drawbacks:
  - sets involve a time overhead for connection/disconnection to or from a set.
  - sets can cause a space overhead if pointers occupy more room than the field.

The choice of the best compromise is left to the database designer.

3. Data Manipulation: Basic Concepts

This part of the manual is devoted to the concept of a dynamic database, in other words, to the dynamic aspects of the IDS/II model. It is in this section where we consider the actions of a changing database which affect the elements of the data structure It introduces the concepts that are necessary for the understanding of the data manipulation functions described further on.

The concepts outlined in the beginning of this section give an overview of the zones involved in IDS/II database activity. The first concept which leads us into this disucssion is the concept of **area**.

The concept of **area** serves to partition database records on a logical as well as physical basis.

Related to the area are **data-base-keys** and **area-keys**, which provide the means of identifying and referencing records.

Record location modes define the technique involved in the physical placement of records in the database.

Secondary keys provide an accelerated access path for the locatation of records in the database

The **index areas** are logical areas where activity on secondary keys is performed.

The **program/DBCS communication zones** are zones where the program and the DBCS exchange control and data information.

Currency indicators are logical pointers that keep track of records recently accessed and allow subsequent manipulation functions to reference them indirectly.

The **set membership class** defines the conditions in which a potential member of a set is actually connected to this set.

The **set selection criteria** are rules which govern the automatic selection of set occurrences when a member is connected to this set.

The **set ordering criteria** are rules which govern the order of logical insertion of members into a set occurrence.

The **no duplicates control** rules are a group of DCBS rules which forbid records with duplicate values in certain fields within an area or a set occurrence.

Validity checks on data items are checks based on DCBS rules which forbid the presence within the database of records whose field contents do not satisfy certain conditions.

Two of the mechanisms mentioned above are specified in the DML language of IDS/II: **program/DCBS communication zones** and **currenty indicators**. The other concepts give rise to descriptive specifications in either DDL or DMCL, but these are basically procedural descriptions.

A brief summary of retrieval, update, and control functions are given below. These functions will be more fully explained in subsequent sections.

FIND	locates a record in the database.
GET	moves the contents of a record from the database to the User Working Area.
ACCEPT	reads information from the currency indicators or from the schema DMCL.
DB Condition (IF)	performs tests of database conditions.
STORE	creates a record in the database.
MODIFY	modifies the contents and/or membership of a record.
ERASE	removes a record from the database.
CONNECT	connects a record to a set occurrence.
DISCONNECT	disconnects a record from a set occurrence.
READY	makes an area available for processing.
FINISH	makes an area unavailable for processing.
USE	centralizes the processing of data-base-exceptions.

3.1 AREAS IN THE DATABASE

3.1.1 Relationship between Areas and Records

Areas or **realms** are storage containers for records. There is no area-type and each area has a name. **Area** is the DDL term and **realm** is the COBOL term.

The relationship between records and areas is described by the following rules:

- A record occurrence must belong to one and only one area.
- A record-type may have occurrences in one or several areas.
- An area may contain occurrences of different record-types.

As a result of these rules, the following are true:

- Areas are **disjointed subdivisions** of the database, each one separate. (See Figure 3-1.)
- A set occurrence may be contained in one area (all tenants are in this area) or may span areas (tenants are scattered in several areas).

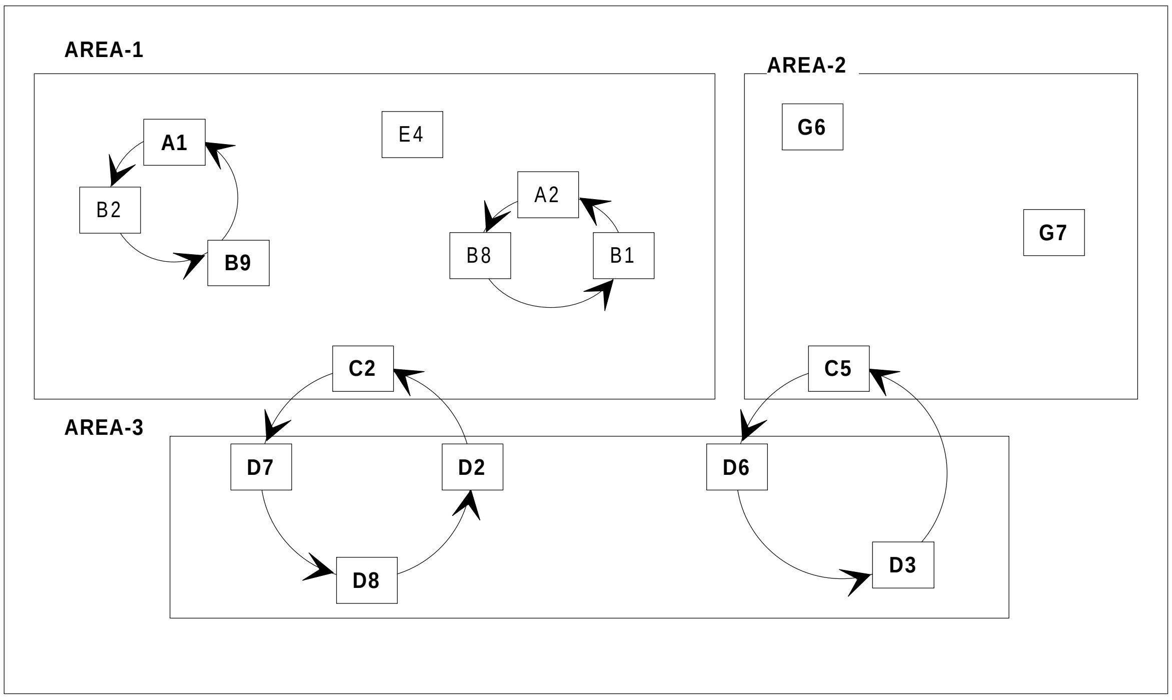


Figure 3-1. Database as a Group of Disjointed Areas

3.1.2 Using Areas

The concept of **areas** becomes necessary when we pass from the static database structure to the actual operation of the database. Areas make more efficient use of the database possible for the following practical reasons:

- Programs can open (READY) only the areas they require for specific operations, which avoids keeping the whole database on line.
- The area is a convenient unit for copy, save, and restore operations.
- The database administrator can control the assignment of records to areas based on those records' frequency of use.
- The area can constitute a level of concurrent access control between several users of the database. When opening an area, the program may request an exclusive or shared usage of that area and specify whether it intends to work in retrieval mode or update mode on that area.
- The area may also be used to distinguish the placement of records according to different levels of security. As explained below, areas correspond to UFAS files in which each area can be submitted to the file access rights provided by the Catalog.

3.1.3 The Storage Area

The storage area is defined in the schema DDL. To each schema area there is a corresponding storage area in the DMCL descriptions which is a UFAS integrated file.

The storage area is the unit used for static space allocation and run-time assignments.

A storage area can be divided into **ranges**, which provide an additional flexibility in the record placement strategy. One range can be defined per record-type in the area. The ranges of different record-types may be identical, disjoint, or may overlap (see Figure 3-2). If a range is not defined, it is assumed to be equal to the area.

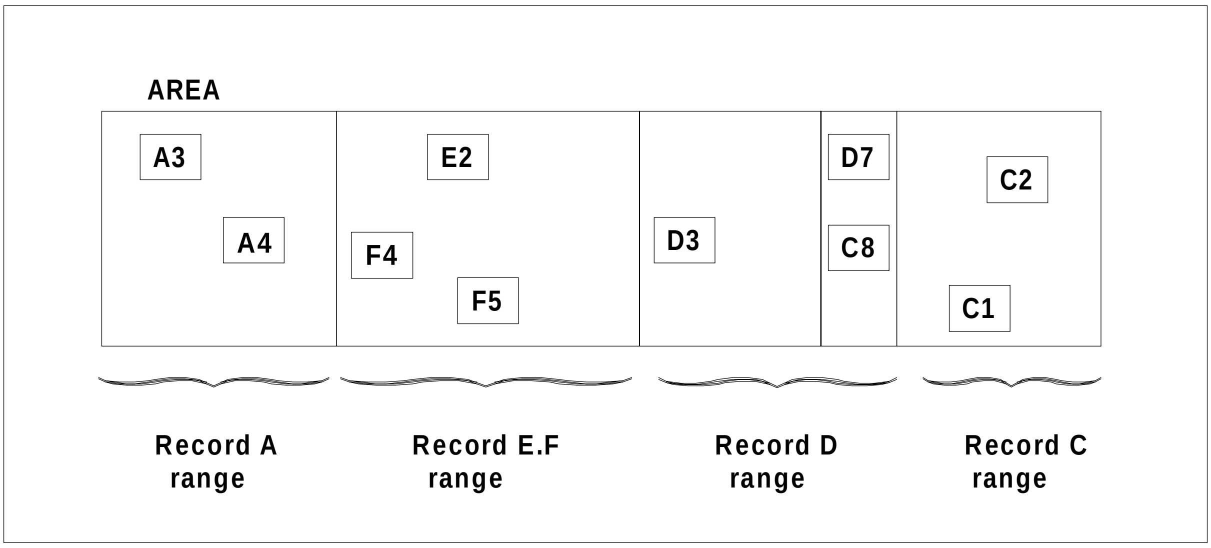


Figure 3-2. Ranges Within Areas

3.2 DATA-BASE-KEY AND AREA-KEY

3.2.1 The Structure of the Storage Area

Each storage area is a UFAS integrated file divided into **pages** of equal length, each page having the same potential number of **lines** (records).

The page is the **unit of transfer** between the disk and the DBCS buffers.

The line is the container for one record. Its size is variable according to the type and length of record it currently contains.

The number of pages, the page length, and the number of lines per page may vary between areas (see Figure 3-3).

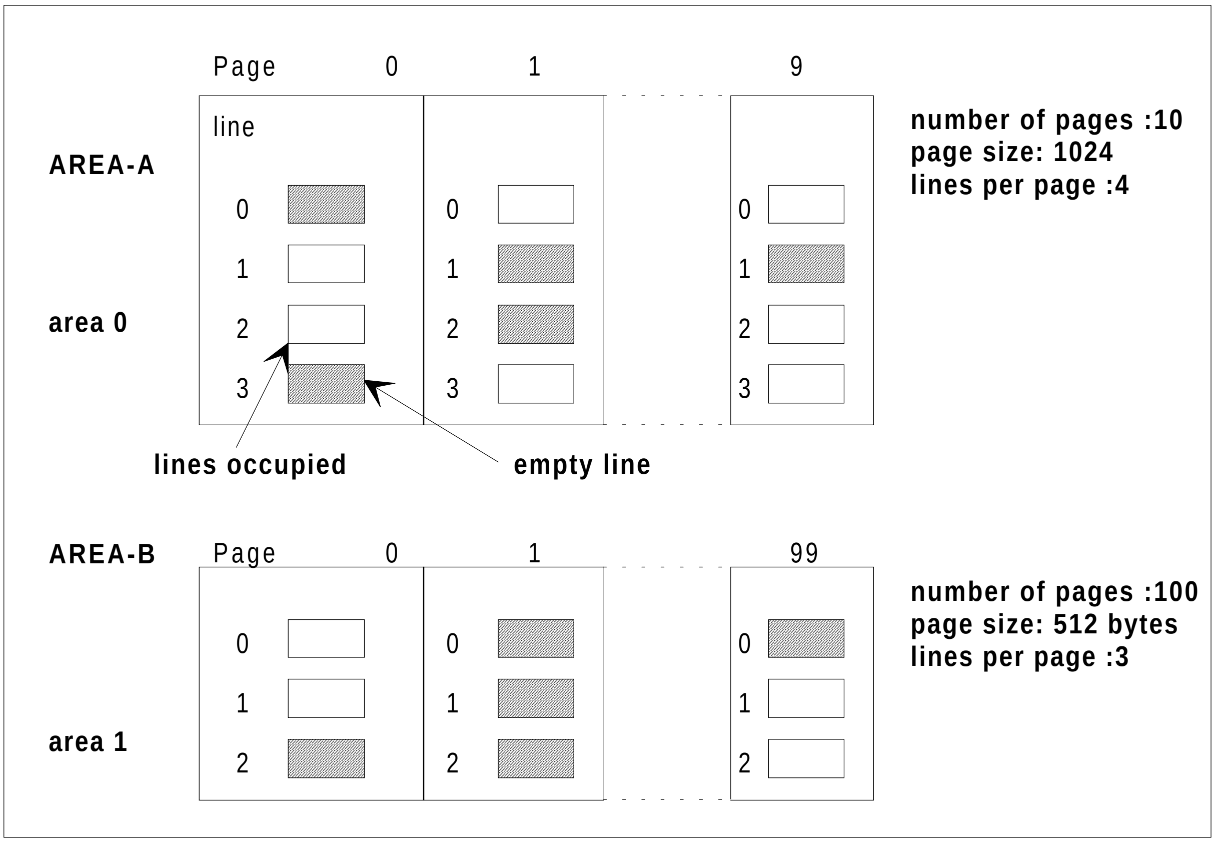


Figure 3-3. Physical Storage in the Database

3.2.2 Assignment of a Record Address

When a record is stored in the database, it is placed at a logical address whose components are:

- an area number
- a page number within the area
- a line number within the page

This logical address is called its **data-base-key**. The data-base-key uniquely identifies the record within the database, independently of its record-type or participation in any set-type. This identifier is the one used internally in the set pointers.

The **data-base-key of a record remains constant during its life** (except in case of CALC record migration). When the record is deleted, its logical address becomes free again and can be reassigned to a new record.

The local address in an area, consisting of the **page number** and the **line number**, is called **area-key**. It is the record address of that area.

3.2.3 Data-Base-Key Format as seen by a COBOL Program

COBOL data items containing data-base-keys must be declared with "USAGE IS DB-KEY".

In IDS/II, the data-base-key is designed as a positive "FIXED BIN 31" item. Such fields can be used without restriction in the MOVE and COMPARE statements of COBOL and in COBOL algebraic operations.

The internal structure of the data-base-key must be known to the programmer if he has to code a placement algorithm for DIRECT records. It is illustrated by Figure 3-4.

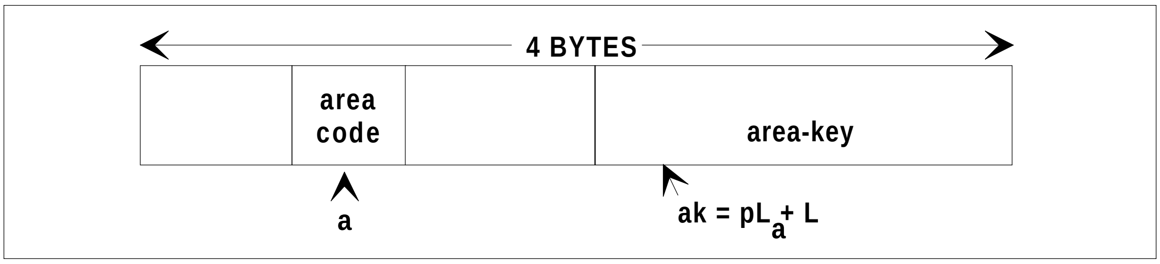


Figure 3-4. Data-base-key Format in a COBOL Program

Meaning of symbols in the figure:

a	- the area code number (starting from 0)
p	- the page number (starting from 0)
La	- the number of lines-per-page of area a
ak	- the area key
L	- the line number (starting from 0)

The size and position of the area code zone are constant for a given database. Section III of the *IDS/II Reference Manual Volume 1* describes how to determine the position of the area code zone so as to minimize the size of the set pointers.

The area-key zone is always right-justified but its size depends on the area, since the number of pages and number of lines-per-page may be different between areas.

In order to facilitate the computation of data-base-keys, IDS/II provides extensions to the ACCEPT verb which enable the programmer to read the following information from the schema:

- the number of lines-per-page of an area
- the number of pages of an area or of a record range within an area
- the minimum data-base-key of an area (for example, k=0) or of a record range within an area.

3.3 RECORD LOCATION MODES

DDL is used to define the following for each record-type:

- the areas where the record can be stored WITHIN clause, and, if there is a choice, the parameter that indicates the area selected (AREA-ID)
- the method of assigning an area-key to the record once the area has been determined (LOCATION clause)

The syntax of these two clauses is given below:

```
                 { DIRECT data-base-parameter-1               }
                 {                                            }
LOCATION MODE IS { CALC USING {data-base-identifier-1} . . .  }
                 {         DUPLICATES ARE [NOT] ALLOWED       }
                 { VIA set-name SET                           }

       {{ANY AREA         } [AREA-ID IS data-base-parameter-2] }
WITHIN {{{area-name} . . . }                                    }
       {AREA OF OWNER                                          }
```

Records can be stored into an area according to one of three modes, which are presented in summary form below. Further details are supplied in the descriptions of the STORE command.

1) LOCATION IS DIRECT.

   Working in this mode, the programmer provides the **area-key** (or data-base-key) of the record in "data-base-parameter-1". If the line is already occupied or if there is not enough room in the page to assign the empty line to the record, the DBCS places the record in the next available location. This location mode must be avoided since it is not compatible with an automatic reorganization of the database by a utility.

2) LOCATION IS CALC.

   In this mode, the DBCS computes the area-key of a record by using a hashing calculation on the contents of the elementary field "data-base-identifier-1" (also called the "CALC-key") of that record. The result of that calculation is divided by the number of **buckets** of the area. (A **bucket** is a container composed of a given number of contiguous pages and is defined in the DMCL.) The result of the division determines the page where the record should be placed. Records which randomize to the same bucket are called **synonyms**. The DBCS chains these together and manages the possible overflow onto subsequent pages. (There exist as many potential "CALC" chains as buckets). The **DUPLICATES** option allows the user to accept or reject a new record having the same type and CALC-key value as a record which is already present in the area. The **DUPLICATES NOT** check applies only to the area involved in a STORE action.

3) LOCATION IS VIA.

In this mode, the placement of the record is determined by the location of the owner in the set. The result is a **clustering** of set members within a number of contiguous pages. These pages are usually near the owner, if members and owner are in the same range of the same area.

The LOCATION mode of a record-type is the same for all the areas in which it can be located.

3.4 SECONDARY KEYS

3.4.1 The Concept of Secondary Keys

A secondary key is either a data item or a record-type for which an accelerated retrieval access path exists. This path is called an INDEX. The following items and types can qualify as secondary keys:

- a single data item (**a mono-field key**)
- a sequence of data items (**a multi-field key**)
- a record-type determined from a given record (**a mono-record-type key**)
- several record-types (**multi-record-type key**)

  Suppose that the database contains records having the following secondary key values:

  - ADAMS
  - ADLER
  - ASCOT
  - ASHTON
  - ...
  - YALTA
  - YORK
  - ZEUS
  - ZWEIG

For each secondary key value, the **index** will contain a pointer (db-key) to a record in the database (see Figure 3-5).

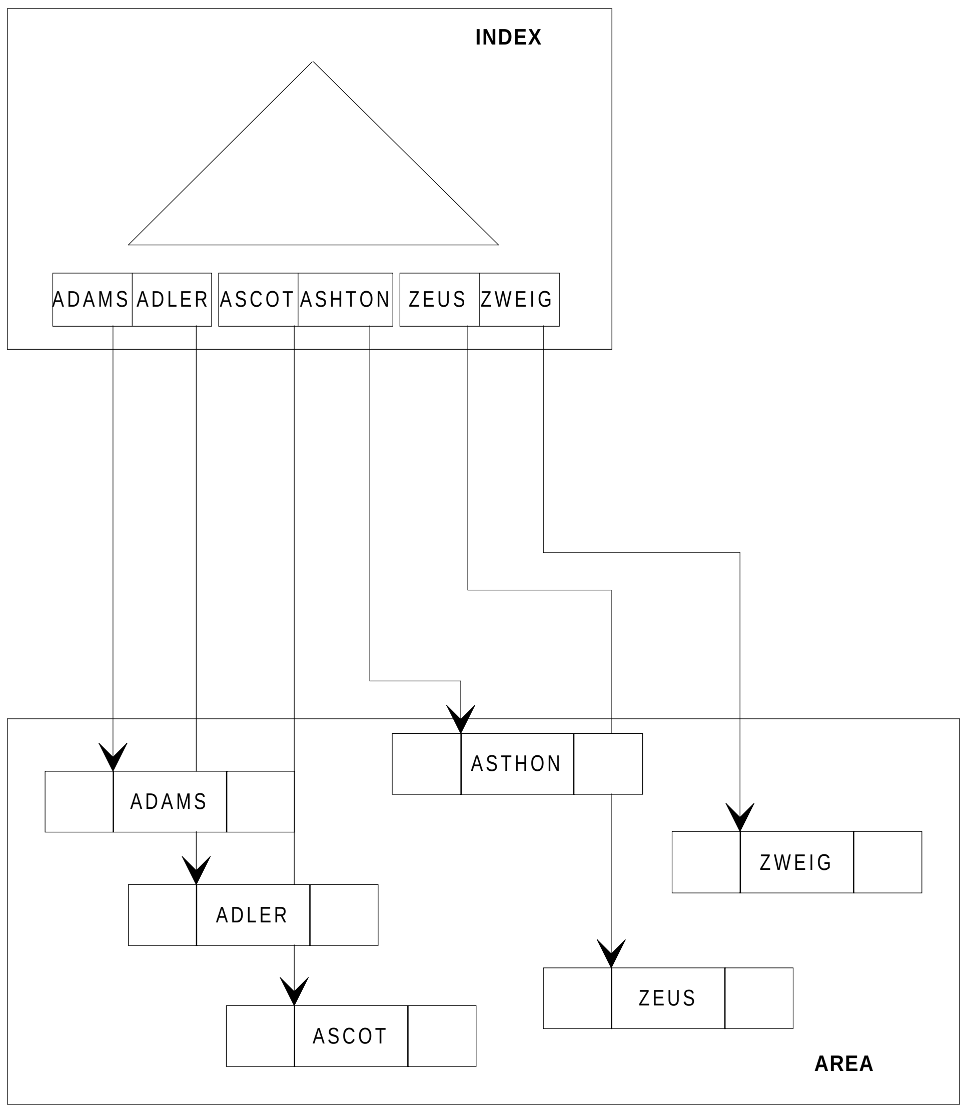


Figure 3-5. The Concept of Secondary Keys

The term **secondary** is used because the physical location of a record in the database is not affected by the declaration of a key for that record. The range of a secondary key can encompass the entire database or be restricted to a given area. Duplicate secondary key values can be declared as **allowed** or as **not allowed**.

In order to declare a secondary key in the DDL, one must specify two entries:

- a **RECORD KEY SUBENTRY** for each record which participates in the key. This is to ensure a description of each data item of the record which defines the secondary key.
- a **KEY ENTRY**, to specify the properties of the secondary key.

The DDL syntax of the KEY Subentry is given below:

```
                              {ASCENDING }
KEY NAME IS key-name-1 USING  {          } db-ident-2
                              {DESCENDING}

        [ [ {ASCENDING } ]            ]
        [ [ {          } ]  db-ident-3] . . .
        [ [ {DESCENDING} ]            ]
```

The DDL syntax of the KEY Entry follows:

```
KEY NAME IS key-name

  [       {db-parameter-1}         ]
  [ USING {              } . . .   ]
  [       {db-ident-1    }         ]

  [        {ANY                           } ]
  [ WITHIN {                              } ]
  [        {AREA IDENTIFIED BY db-parameter-2} ]

  [ DUPLICATES ARE { [NOT] ALLOWED } ] .
```

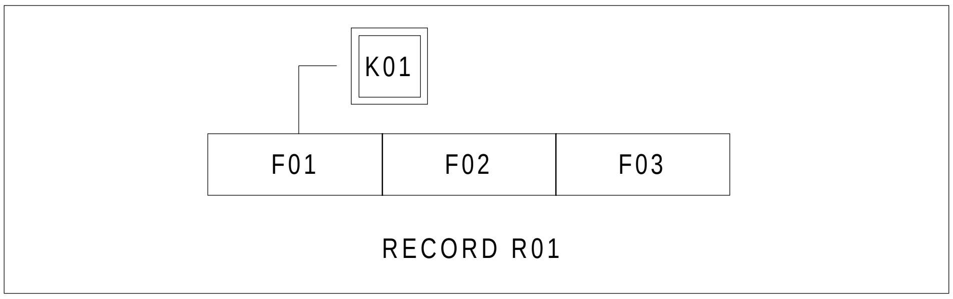
In this manual, a key will be represented by a rectangle, like the one shown below:

\$\

In such illustrations, the key name will appear inside the rectangle. A curving line will connect the rectangle (key symbol) with one or several record types, as shown in the three examples on the following pages:

Example 1:

This example illustrates a **mono-field key** connected to a **mono-record-type key**:

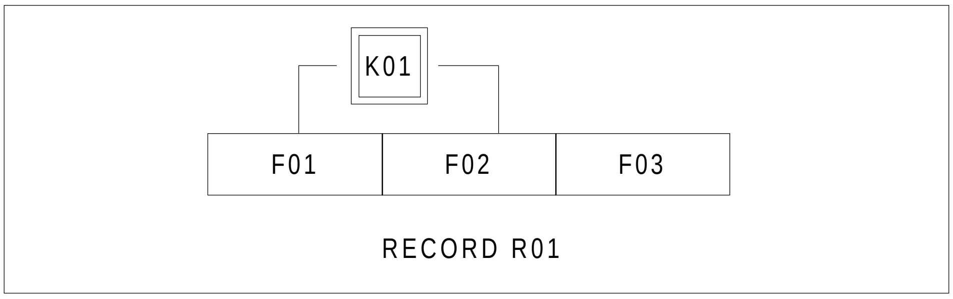


The DDL corresponding to this example is given below:

```
RECORD NAME IS R01
KEY NAME IS K01 USING DESCENDING F01.
        ...
01      F01 TYPE IS CHAR 5.
01      F02 TYPE IS SIGNED BINARY 31.
01      F03 TYPE IS UNSIGNED UNPACKED DECIMAL 9.


        ...

KEY NAME IS K01
    DUPLICATES ARE NOT ALLOWED.
```

Example 2:

This example shows **multi-key fields** connected to a **mono-record-type key**:

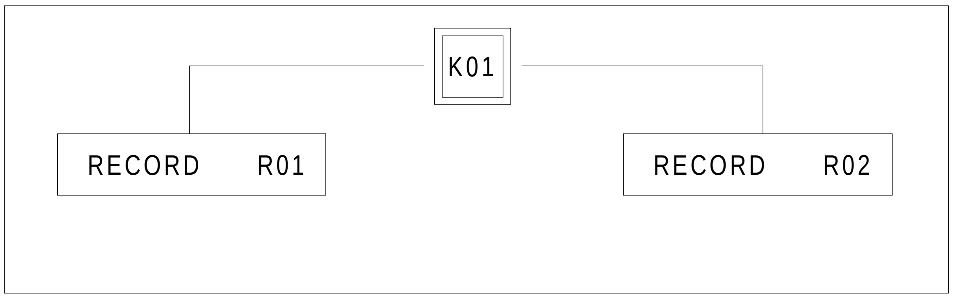


The DDL corresponding to this example is given below:

```
RECORD NAME IS R01
KEY NAME IS K01 USING ASCENDING F01 DESCENDING F02.
        ...
01      F01 TYPE IS CHAR 5.
01      F02 TYPE IS SIGNED BINARY 31.
01      F03 TYPE IS UNSIGNED UNPACKED DECIMAL 9.


        ...

KEY NAME IS K01
    DUPLICATES ARE ALLOWED.
```

Example 3:

The example below illustrates **multi-record-type key** and **multi fields**:



The DDL corresponding to this example is shown below:

```
RECORD NAME IS R01
KEY NAME IS K01 USING R01-F03 R01-F01.
        ...
01      R01-F01 TYPE IS CHAR 5.
01      R01-F02 TYPE IS SIGNED BINARY 31.
01      R01-F03 TYPE IS UNSIGNED UNPACKED DECIMAL 9.


RECORD NAME IS R02
KEY NAME IS K01 USING R02-F01 R02-F02.
        ...
01      R02-F01 TYPE IS UNSIGNED UNPACKED DECIMAL 9.
01      R02-F02 TYPE IS CHAR 5.


        ...

KEY NAME IS K01
    USING      KEY-PARAM-1  KEY-PARAM-2
    DUPLICATES ARE ALLOWED.
```

3.5 INDEX AREAS

3.5.1 Index Areas and Secondary Keys

Index areas are containers for secondary keys. There is no "index area-type"; each index area is given a name.

The following rules define the relationship between secondary keys and index areas:

1) a secondary key occurrence must belong to one and only one index

2) an index must belong to one and only one index area.

3) an index may contain occurrences of more than one secondary key; if more than one key occurrence exists, all secondary keys in the same index must have identical properties (DUPLICATE ALLOWED or NOT ALLOWED for all keys in that index)

4) an index area can contain only one index

5) index areas are **dedicated areas** of the database, and therefore contain no database records.

3.5.2 Examples of Types of Indexes

3.5.3 A Mono-Key Index with Mono-Record Keys where Range = Entire Database

This is the simplest case. Suppose that we have the record-type R which can be located in any area of the database:

- AR-1
- AR-2

and a secondary key **K** declared on record **R**, implemented in index area **I1**, with the following secondary key values for each occurrence of R:

- R1 -----> K1
- R2 -----> K4
- R3 -----> K3
- R4 -----> K2
- R5 -----> K5

and where the following relationships hold true among these keys: K1 < K2 < K3 < K4 < K5.

The index occurrence diagram for this case is shown in Figure 3-6:

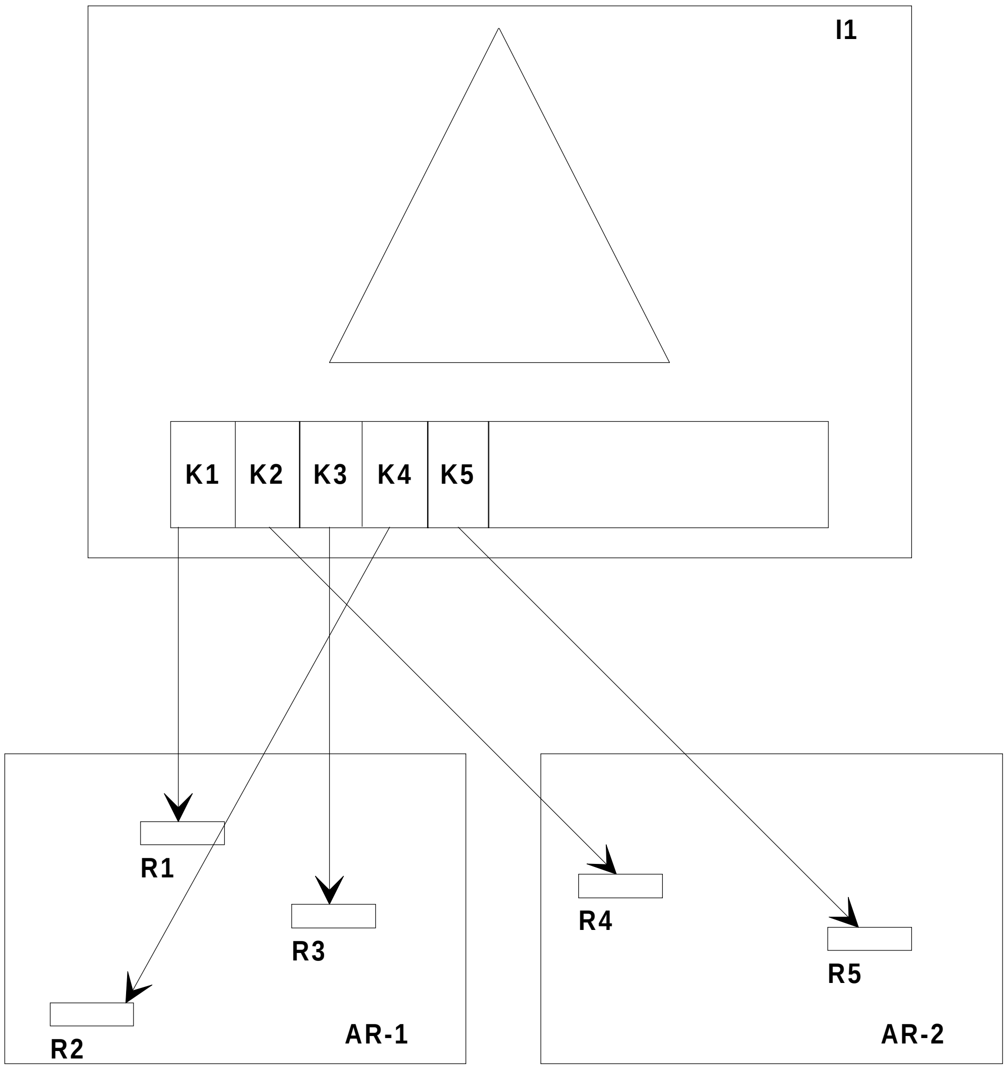


Figure 3-6. A Mono-Key Index with Mono-Record Keys

3.5.3.1 A Mono-Key Index with Mono-Record Keys where Range = Area

Suppose that the range of key **K** is restricted to an AREA, as opposed to the preceding example where the range was the entire database.

The index occurrence diagram for such a case is given in Figure 3-7 below:

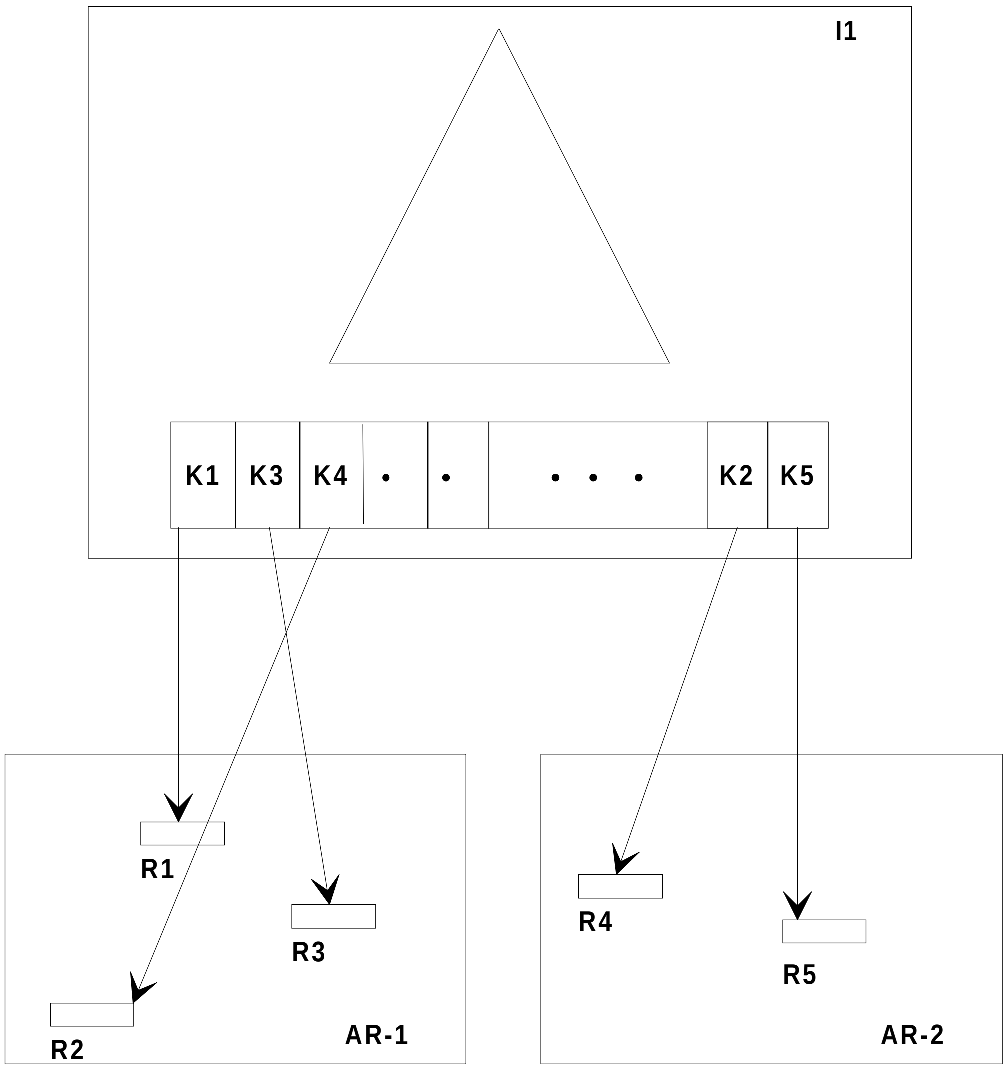


Figure 3-7. A Mono-Key Index with Mono-Record Keys where Range = Area

In the figure above, we see that the values of secondary key **K** which lead to records located in the same area are **grouped together**: keys **K1**, **K3** and **K4**, which lead to records in area **AR-1** are grouped together in that order. The same is true for keys **K2** and **K5**, which lead to records in area **AR-2**.

3.5.3.2 A Mono-Key Index with Mono-Record Keys - DUPLICATES ALLOWED

Suppose now that we have the following secondary key values for each occurrence of R:

- R1 -----> K1
- R2 -----> K3
- R3 -----> K3
- R4 -----> K2
- R5 -----> K3
- R6 -----> K4

where the following relationships hold true: K1 < K2 < K3 < K4. That is, duplicate values are allowed in the secondary keys.
Suppose, also, that the range of K is now the entire database.

Figure 3-8 shows the index occurrence diagram for this case:

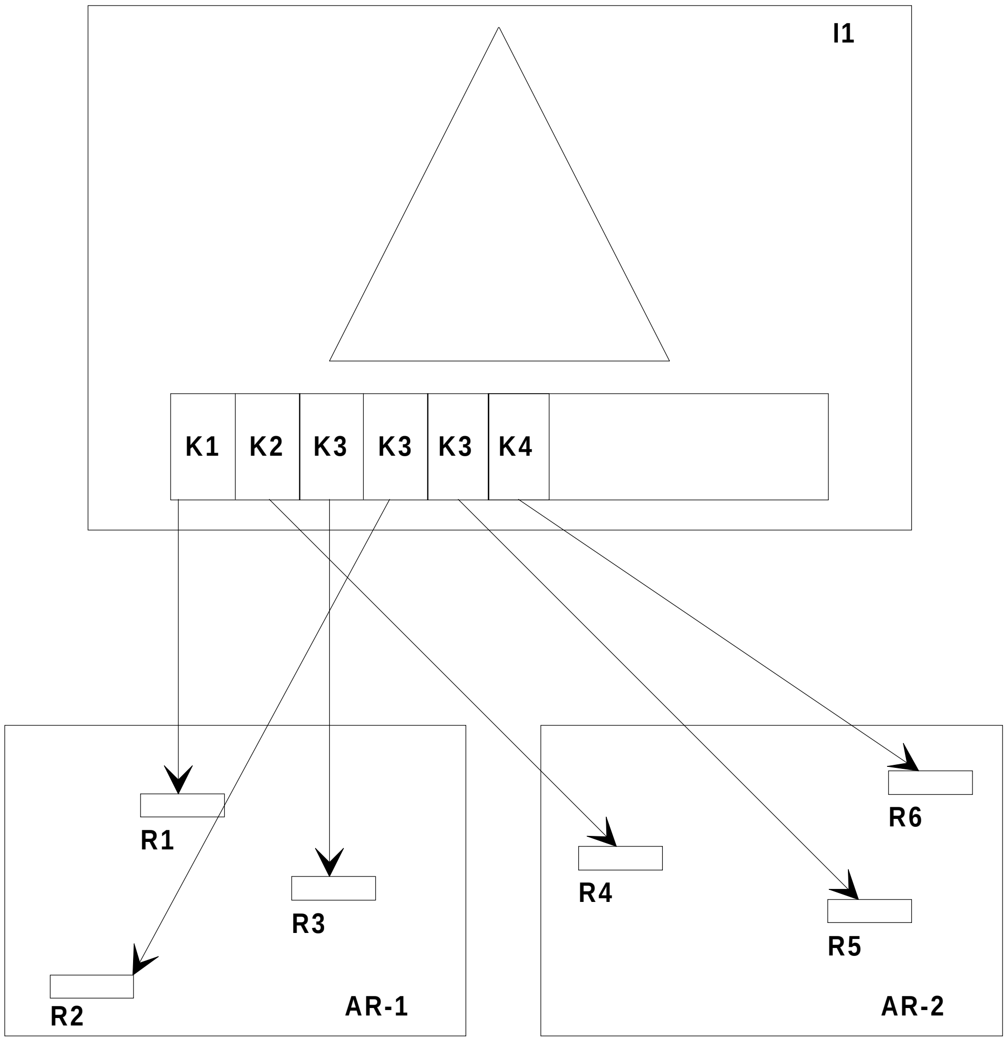


Figure 3-8. A Mono-Key Index with Mono-Record Keys - DUPLICATES ALLOWED

In the case illustrated in Figure 3-8, duplicates values of **K3** are grouped together.

3.5.3.3 A Mono-Key Index with Multi-Record Keys

Suppose that we now have 2 record types, R and T, which can be located in any area of the database:

- AR-1
- AR-2

A secondary key K is declared on records R and T, which is implemented in index area I1, with the following secondary key values for each occurrence of R and T:

- R1 -----> K1
- R2 -----> K4
- R3 -----> K2
- T1 -----> K3
- T2 -----> K5

and with the following relationships: K1 < K2 < K3 < K4 < K5.

Figure 3-9 on the following page, provides the index occurrence diagram for this case.

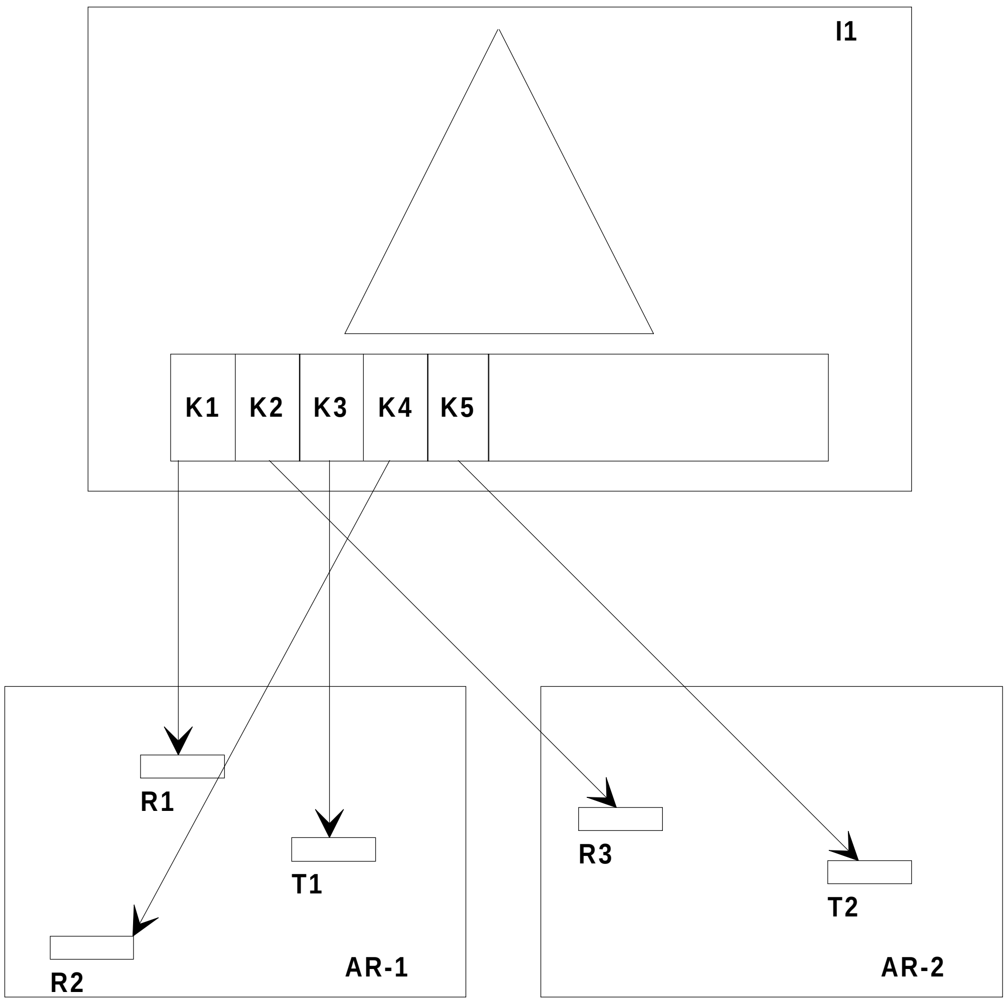


Figure 3-9. A Mono-Key Index with Multi-Record Keys

3.5.3.4 A Multi-Key Index

Suppose that we have two record types, R and T, which can be located in any area of the database:

- AR-1
- AR-2

and suppose that for these records:

- a secondary key, K, is declared on record R
- secondary key, J, is declared on record T

which are implemented in index area I1, with the following secondary key values for each occurrence of R and T:

- R1 -----> K1
- R2 -----> K4
- R3 -----> K3
- R4 -----> K2
- R5 -----> K5
- T1 -----> J1
- T2 -----> J3
- T3 -----> J2

The following relationships are also established among the secondary keys:

a) K1 < K2 < K3 < K4 < K5.

b) J1 < J2 < J3 .

For this case, the index occurrence diagram is provided in Figure 3-10, on the following page:

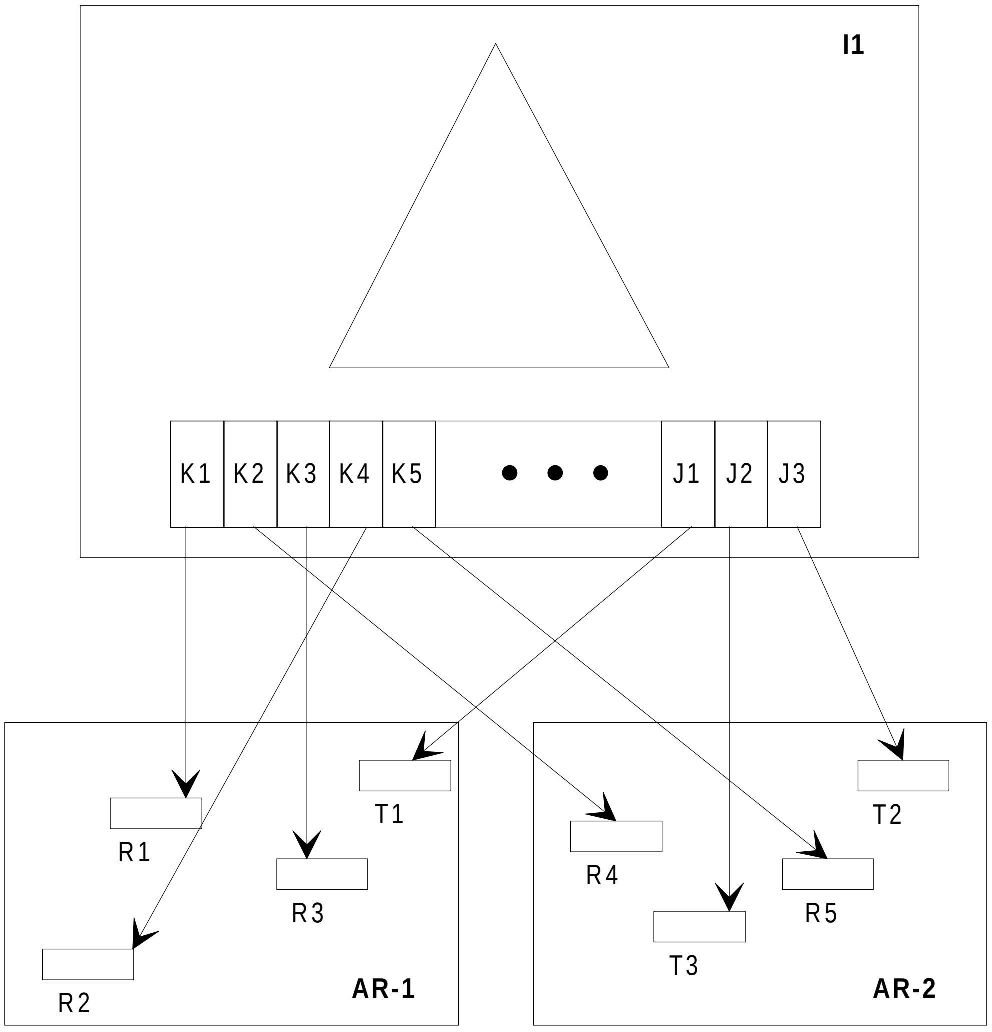


Figure 3-10. A Multi-Key Index

In this instance, secondary key values corresponding to the same secondary key are **grouped together**. In other words, K1, K2, K3, K4 and K5 are grouped according to their secondary key, **K**; the same occurs with J1, J2 and J3, which are grouped according to their secondary key, **J**.

3.5.3.5 Multi Indexes

In the context of the preceding example, one can also choose to have a separate index for each secondary key:

- I1 for secondary key **K**,
- I2 for secondary key **J**

Therefore, the term multi indexes indicates that a separate index is implemented for each secondary key. Figure 3-11 provides the index occurrence diagram which illustrates this situation.

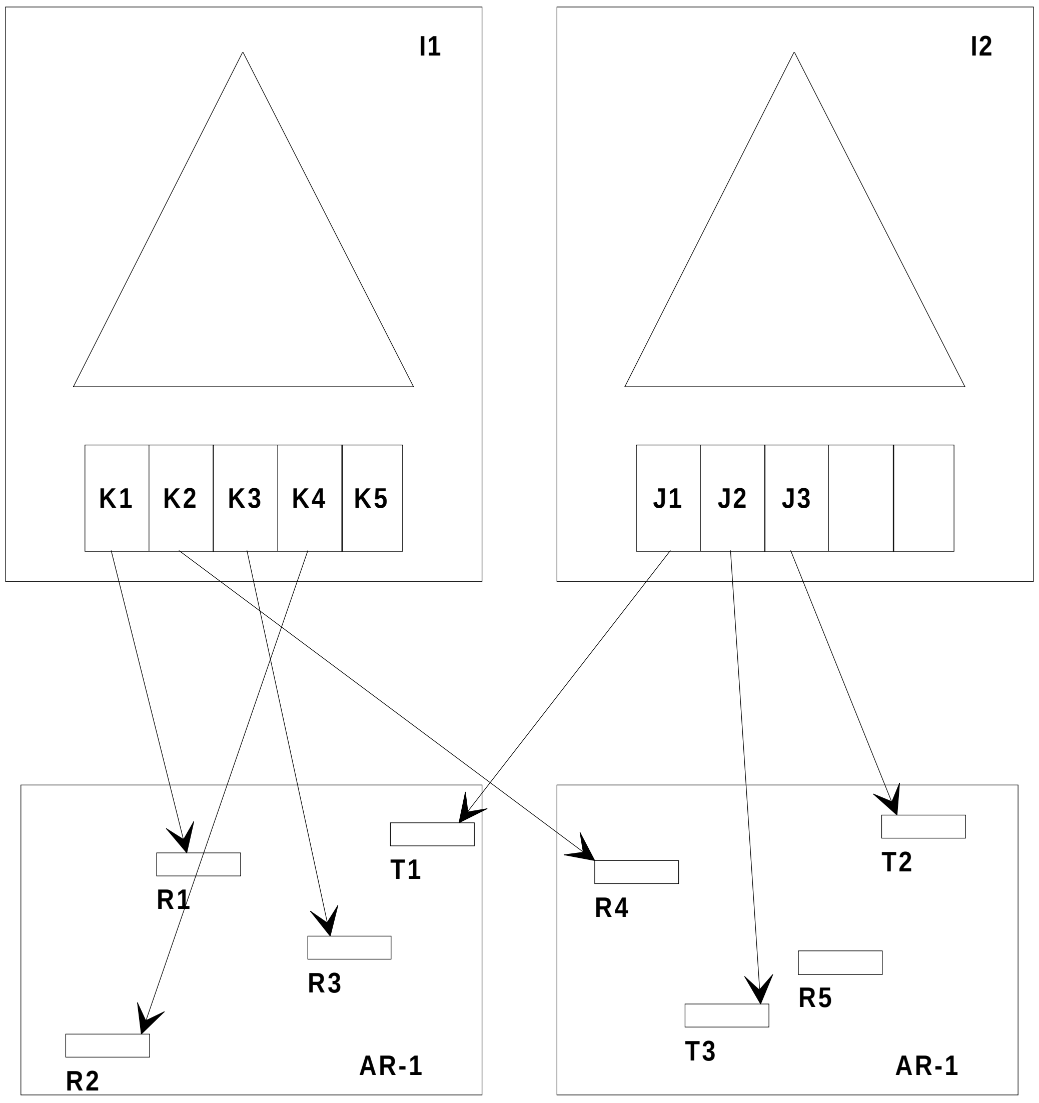


Figure 3-11. Multi Indexes

3.5.4 The Structure of an Index Area

Each index area is a UFAS integrated file which is divided into **pages** of equal length. Each page contains a variable number of **items**.

The page is the **unit of transfer** between the disk and the DBCS buffers.

The following divisions exist in the structure of index areas:

- the **leaf node**, which is a page containing secondary key values, and for each of those key values, a list of all the database keys of database records which match this value.

- the **non-leaf node**, which is a page containing the minimum partial key values that discriminate between child nodes.

- a **child node** is a page which has a parent.

- the **root node**, which is a page which has no parent.

The number of pages, the page length, and the number of items per page may vary between index areas. Suppose that the database contains records having the following secondary key values:

- ADAMS
- ADLER
- ASCOT
- ASHTON
- ...
- YALTA-YORK
- EUS
- ZWEIG

The index structure for these records is shown in Figure 3-12, on the following page:

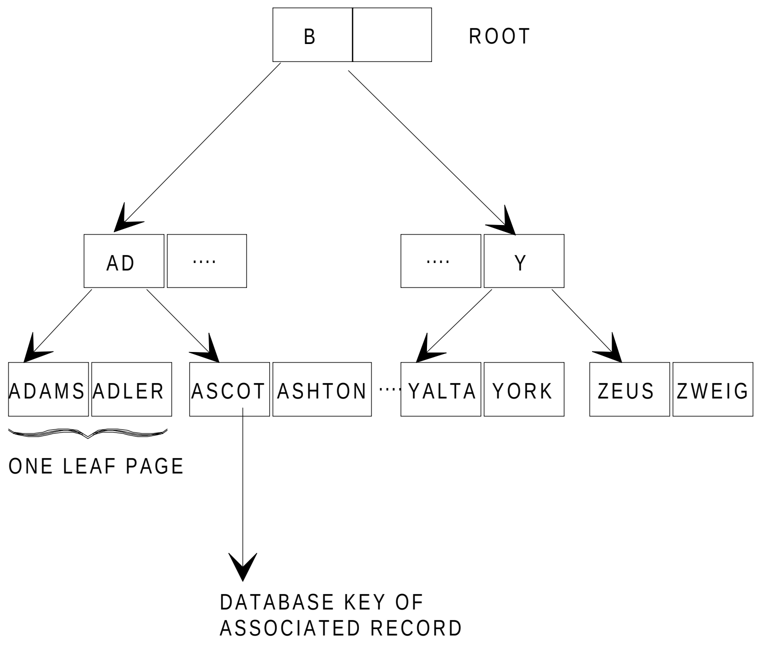


Figure 3-12. Page Structure in an Index

In the figure above, the **root** node contains the single value **B**. Associated to the value **B**, we have 2 pointers:

- one pointing to the child node on the left, which has values less or equal to **B** (value **AD** ...);

- and another pointing to the child node on the right, which has values greater than **B** (value **Y** ...)

The child node containing the single value **AD** has 2 pointers:

- one pointing to the child node on the left, which has values less or equal to **AD** (values **ADAMS** and **ADLER**); the comparison is truncated to the length of AD.

- one pointing to the child node on the right, which has values greater than AD (values **ASCOT** and **ASHTON**)

The child nodes at the lowest level are defined as **leaf pages**, since they contain the full secondary key values which are stored in the database. The leaf page on the right side, mentioned above, contains 2 secondary key values:

- ASCOT
- ASHTON

and it also contains, for each of those secondary key values, a pointer to the associated database record matching these values.

In order to retrieve the record which matches the secondary key value **ASCOT**, the DBCS accesses the following pages:

- the **root node page**
- the **child node page** on the left, associated with the value **B**
- and the **child node page** on the right, associated with the value **AD**

In the current leaf page, the value **ASCOT** (which matches the secondary key) is then associated to the database key of the record found (see Figure 3-13).

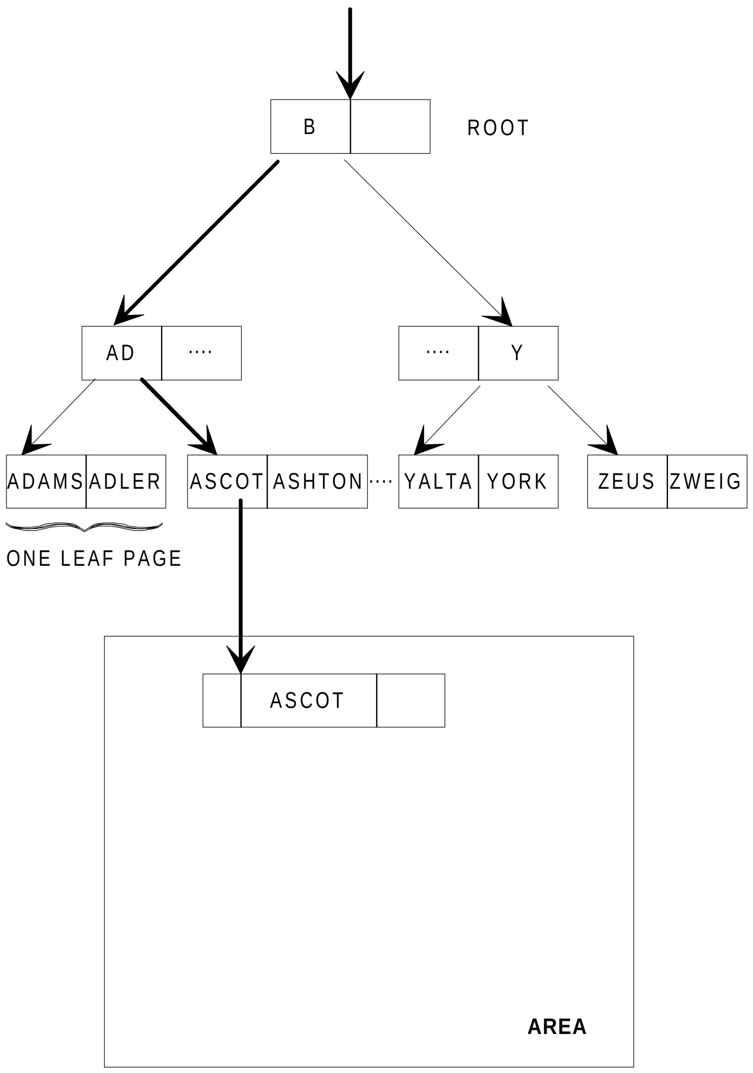


Figure 3-13. Retrieval Path For A Given Key Value

A page contains items in a compacted form, which can be either **full** secondary key values or **partial** secondary key values. The common prefix value of all items of a page is stored only once in the page.

Suppose that we have the following values stored in an index page:

- ADA
- ADAMS
- ADAMS
- ADONIS
- ADVANCE
- ADVICE

Since these values share the common prefix substring **AD**, this prefix is stored only once in the page (see Figure 3-14).

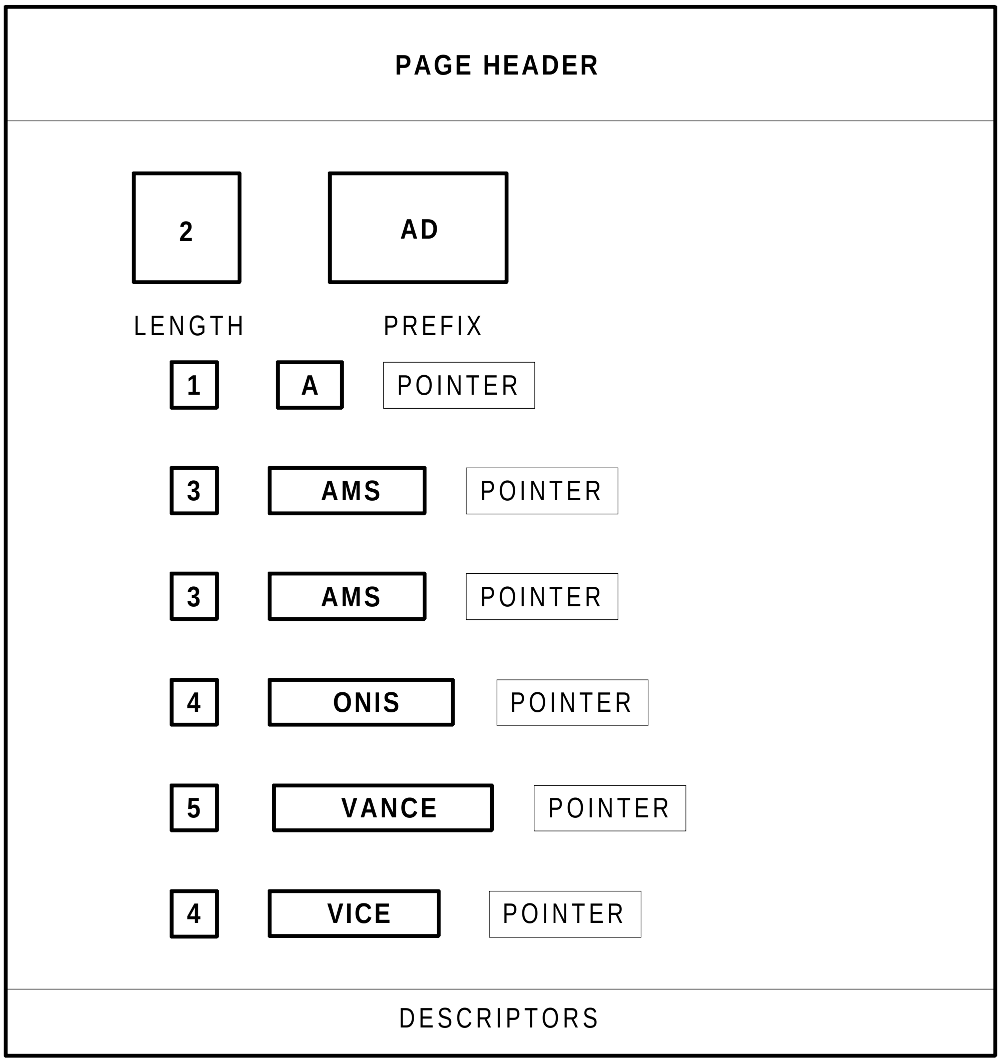


Figure 3-14. Structure of an Index Page

We can distinguish four parts in the general structure of an index page:

1) the page header

2) the variable prefix

3) variables related to the keys, which are associated either to:

    1 : a pointer to the child node for non-leaf pages, or

    2 : a record database key for leaf pages.

4) the page descriptor

3.5.5 Using Index Areas

The concept of an index area is essential to the data structure **if at least one secondary key is declared in the Schema DDL**.

Programs are not authorized to open (READY) an index area. The DBCS controls access to index areas. In this context, the DBCS determines whether exclusive or shared usage of index areas is to be established depending on how corresponding storage areas are utilized.

3.5.6 Detachable Indexes

Unless instructed otherwise, the DBCS automatically updates every index which is impacted during the activities of a data manipulation verb. However, for performance reasons, the user can detach the index from update operations, and defer modifications on the index to a later time. Indexes in IDS/II are therefore **detachable**, since automatic updating can be deferred, and the index can be **detached** from update operations.

An index is termed **detachable** when the user has specified that there be no automatic updating of the index during on-line manipulations by DML verbs. The user must specify this option in the Schema DMCL by using the ON-LINE UPDATE NOT MANDATORY clause.

If an index remains in automatic on-line update, it is called **attached index**

The user can, therefore, **detach** a given index at run-time by using the NO ON-LINE UPDATE idsopt Clause. If the user does not make this choice, the default option is assigned and the index remains attached, allowing on-line update of the index.

It is the user's responsibility to ensure that an index which has been detached remains consistent with the database once updates have been made on database records. To ensure consistency, the user has access to the BUILD KEY utility, which is a tool for reconstructing an index based on the modified database.

Figure 3-15, below, shows the retrieval path used for a record when the index is attached. To retrieve a record in these circumstances, the DBCS performs two operations:

1) it retrieves a secondary key value from the index and the associated database key,

2) it updates certain database currencies so that they point to the database key found by the retrieval action.

For this retrieval operation, the DBCS **does not access** the database.

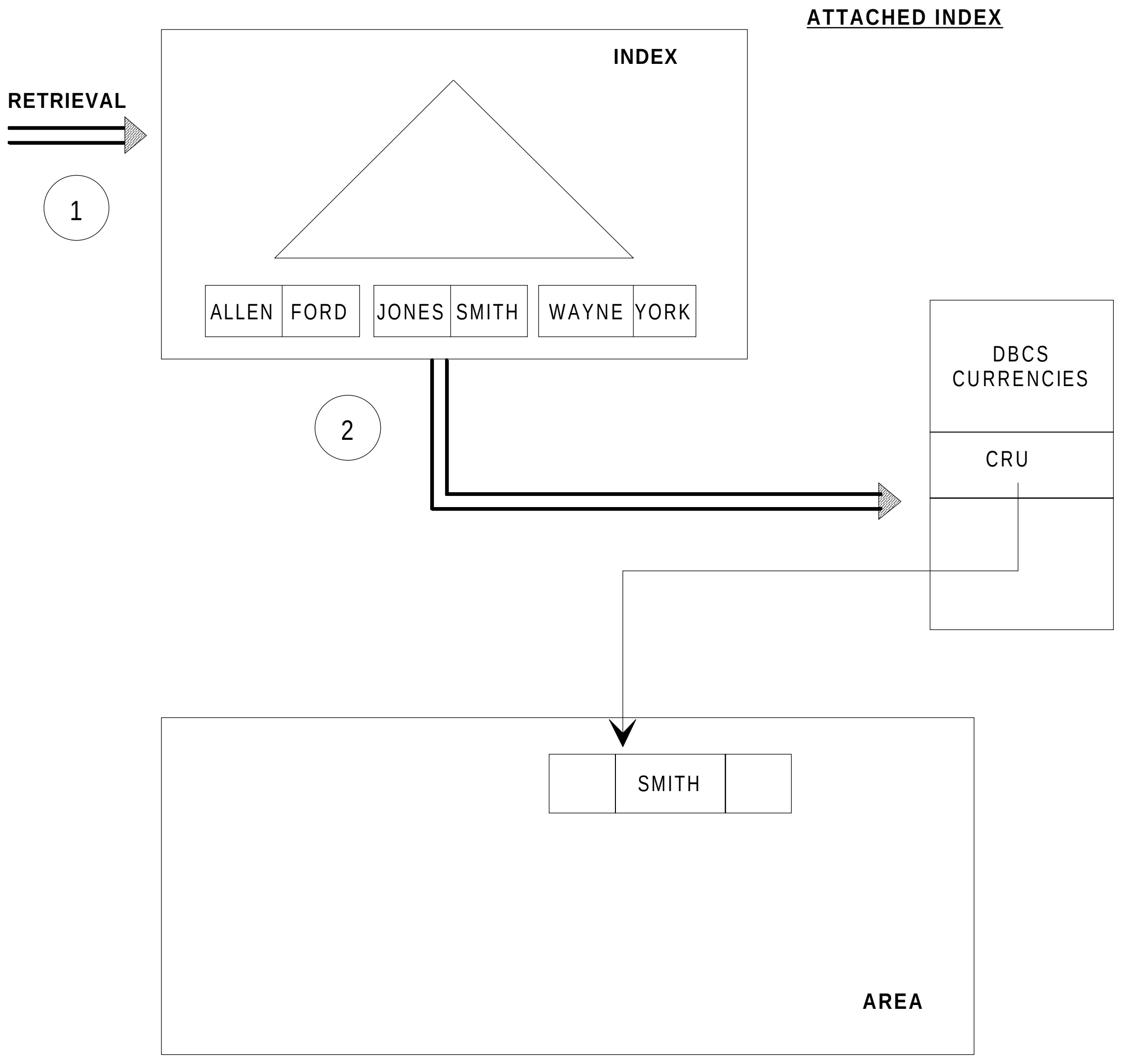


Figure 3-15. Retrieval Path using an Attached Index

Figure 3-16 shows the update path used for a record when the index is **attached** (on-line update). To update a record in these circumstances, the DBCS performs three operations:

1) it updates the database

2) it updates the index

3) it updates certain database currencies

For this update operation, the DBCS **does** actually **access** the database.

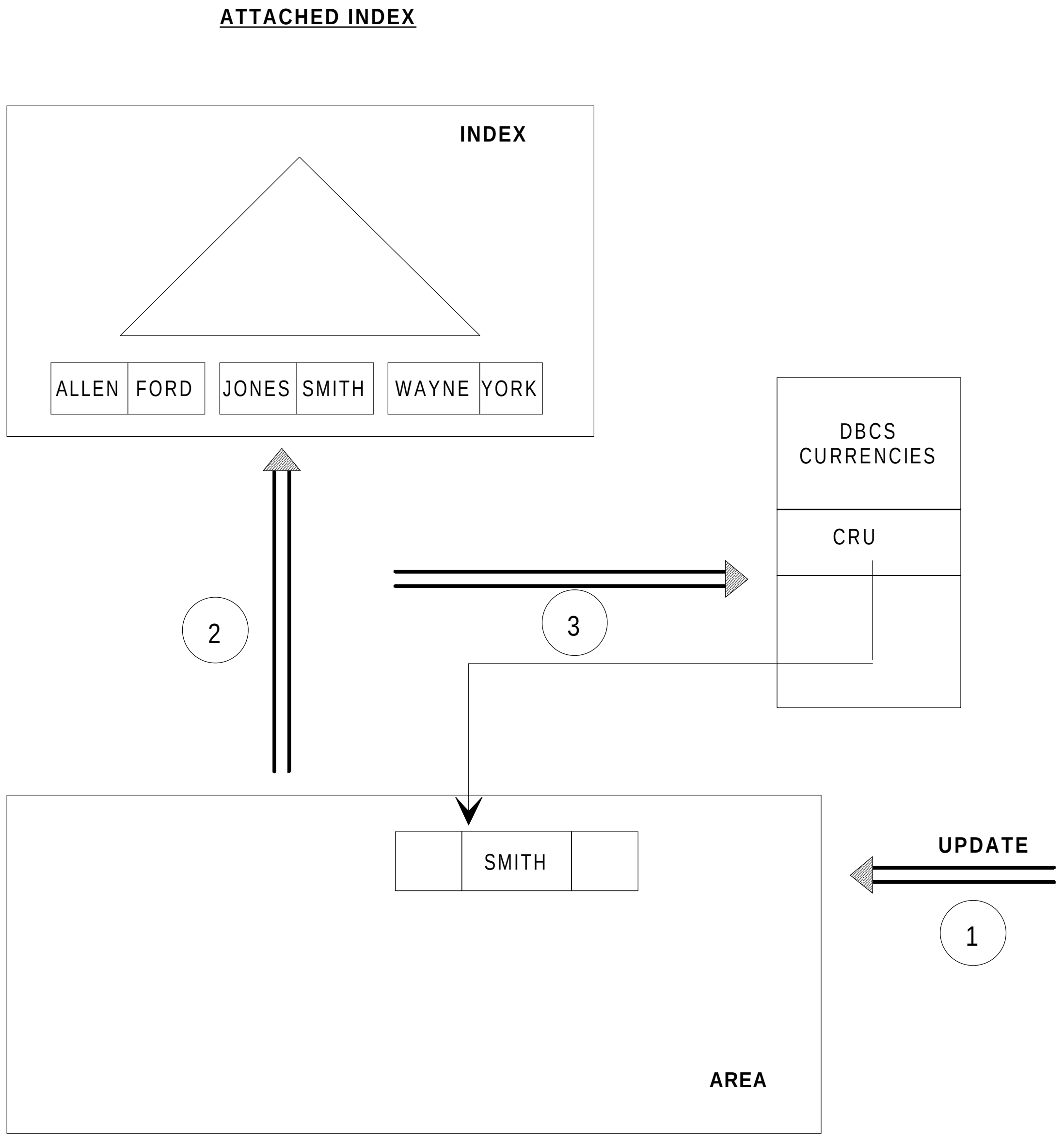


Figure 3-16. Update Path with an Attached Index

Figure 3-17 shows the update path used for a record when the index is **detachable**. To update a record in these circumstances, the DBCS performs two operations:

1) it updates the database

2) it updates certain database currencies

For this update operation, the DBCS **does not access** the index.

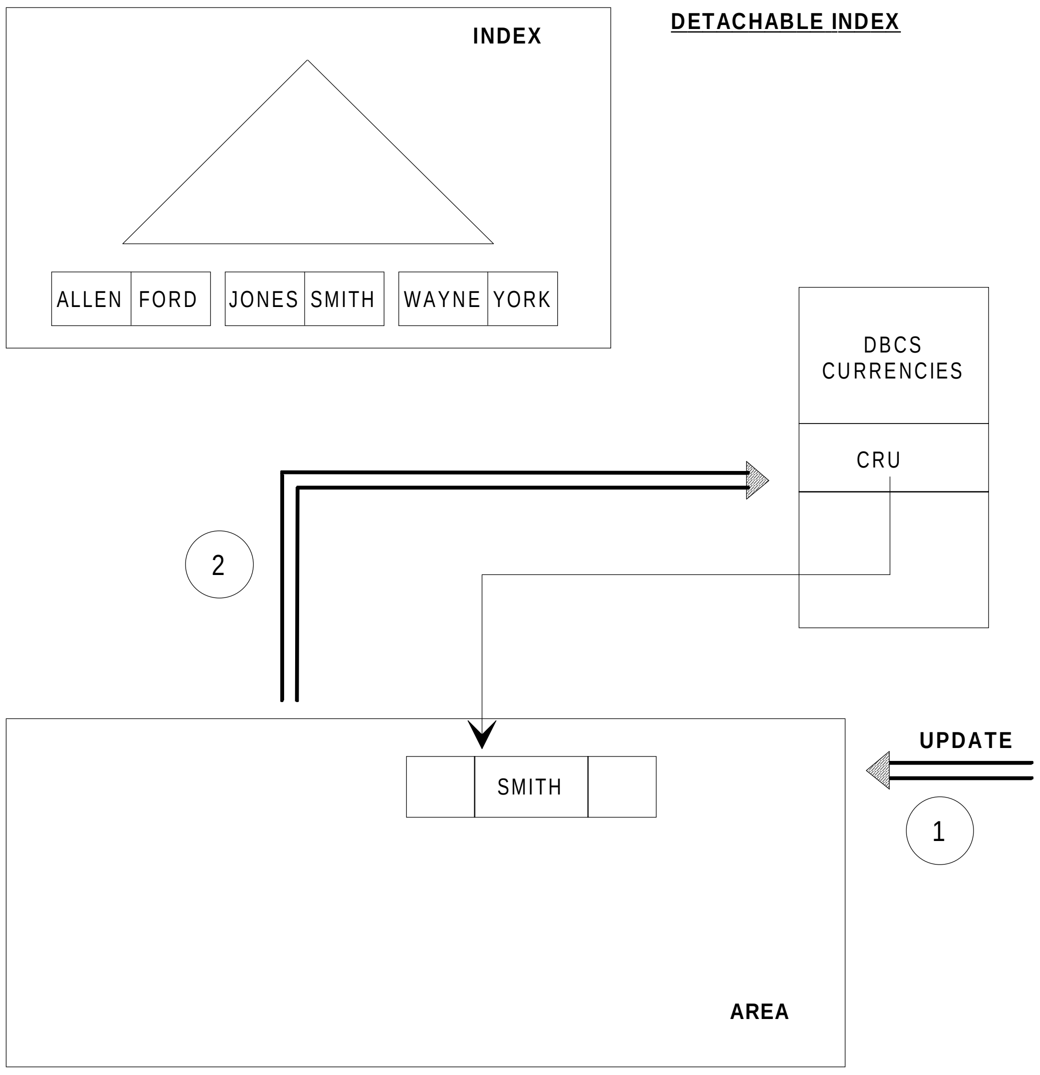


Figure 3-17. Update Path with a Detachable Index

Figure 3-18 shows the retrieval path for a record when the index is **detachable**. To retrieve a record in these circumstances, the DBCS performs three operations:

1) it retrieves a secondary key value from the index and also retrieves the associated database key value.

2) it performs consistency checks on the database

3) it updates certain database currencies.

For this retrieval operation, the DBCS **does access** the database to ensure that the index and the database are consistent.

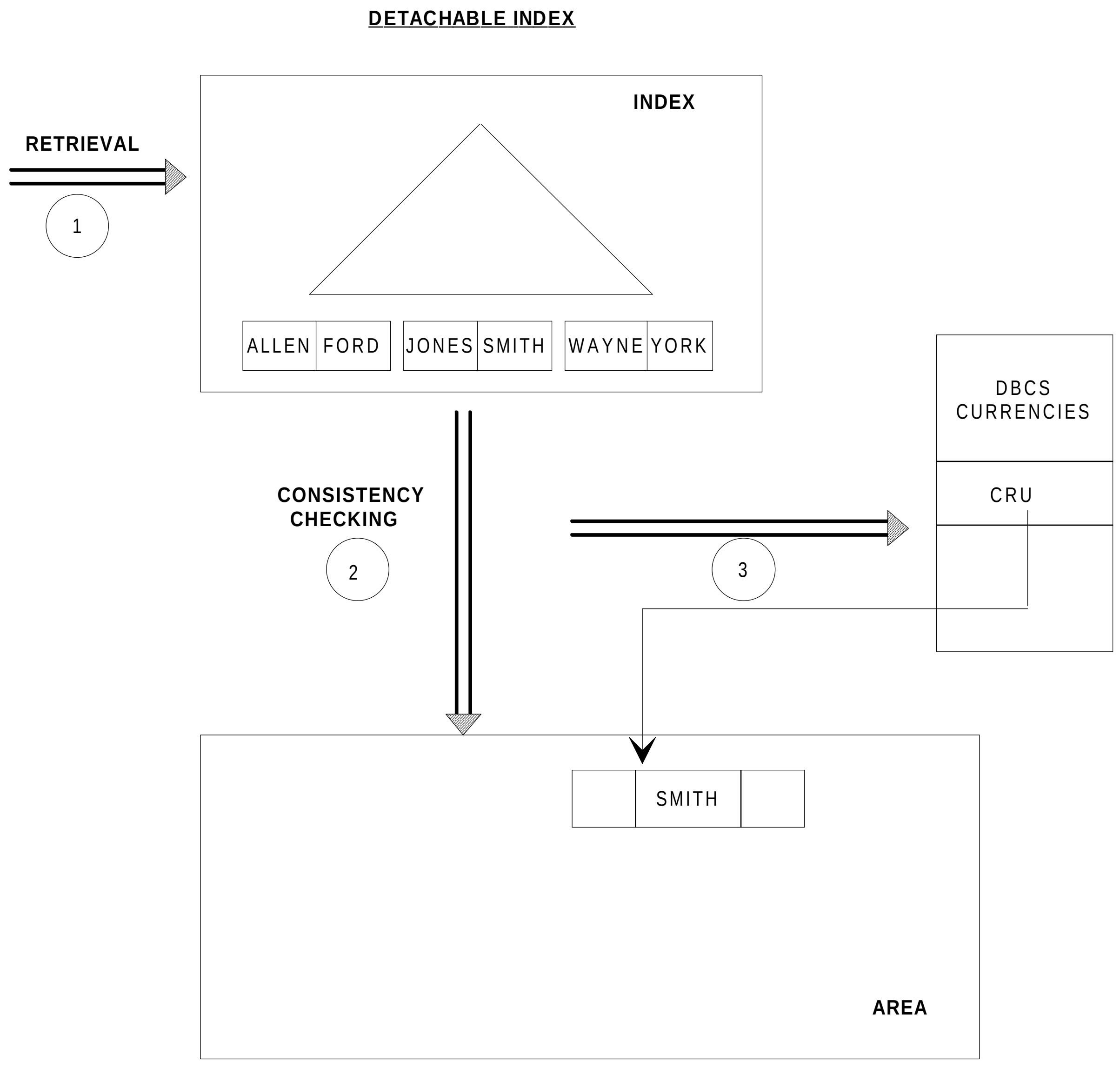


Figure 3-18. Access Path for Record Retrieval with a Detachable Index

3.6 COMMUNICATION ZONES BETWEEN THE COBOL PROGRAM AND THE DBCS

A number of communication zones exist between the COBOL program and the DBCS which ensure the correct operation of DML commands (see Figure 3-19,on the following page). These communication zones can be divided into the following four categories:

1) zones defined in the schema and subschema

2) zones defined by the COBOL programmer in the UWA

3) database special registers

4) currency indicators

These communication zones are described in the subsections below.

3.6.1 Zones Defined in the Schema and Subschema

At compilation time, these zones which are defined in the schema and suschema, are inserted into either the WORKING-STORAGE or LINKAGE section. In the following subsections, the WORKING-STORAGE or LINKAGE is referred to as the USER WORKING AREA (UWA).

These communication zones include record-descriptions and data-base-parameters, and are described below.

1) **Record-descriptions**: these are copied into the corresponding **PICTURE** and **USAGE** clauses of COBOL following the translation of the **TYPE** Clauses of the DDL. For example "TYPE IS CHARACTER 10" becomes "PIC X(10)".

   The record in the UWA is identical to the **data** portion of the record in the database. The record in the database contains an additional **set pointer** which is not visible to the program.

NOTE: An **identifier** which is referenced in the DDL or DML for the purpose of run-time comparisons may designate an elementary field in the UWA record as well as its counterpart in a database record. The semantics of the verb specify whether the identifier is referencing the UWA record or the database record.

2) **Data-base-parameters**: these are referenced in certain DDL clauses such as AREA-ID IS ..., DIRECT ..., EQUAL TO ..., etc. Data-base-parameteres are generated at compilation time in the UWA as fields of a pseudo-record with the name **DB-PARAMETERS**. The data description of these fields depends on the nature of the data-base-parameter, for example, "PIC X(30)" for a realm-name, "USAGE IS DB-KEY" for a data-base-key, etc.

3.6.1.1 Rules for the Use of DML Commands

The programmer is responsible for the initialization of the following in the UWA:

- the complete record in the case of a STORE
- either the complete record or only certain fields in the case of a MODIFY
- the record fields required for comparisons in certain FIND operations
- the CALC-key item(s) in the case of a FIND ANY record-name.

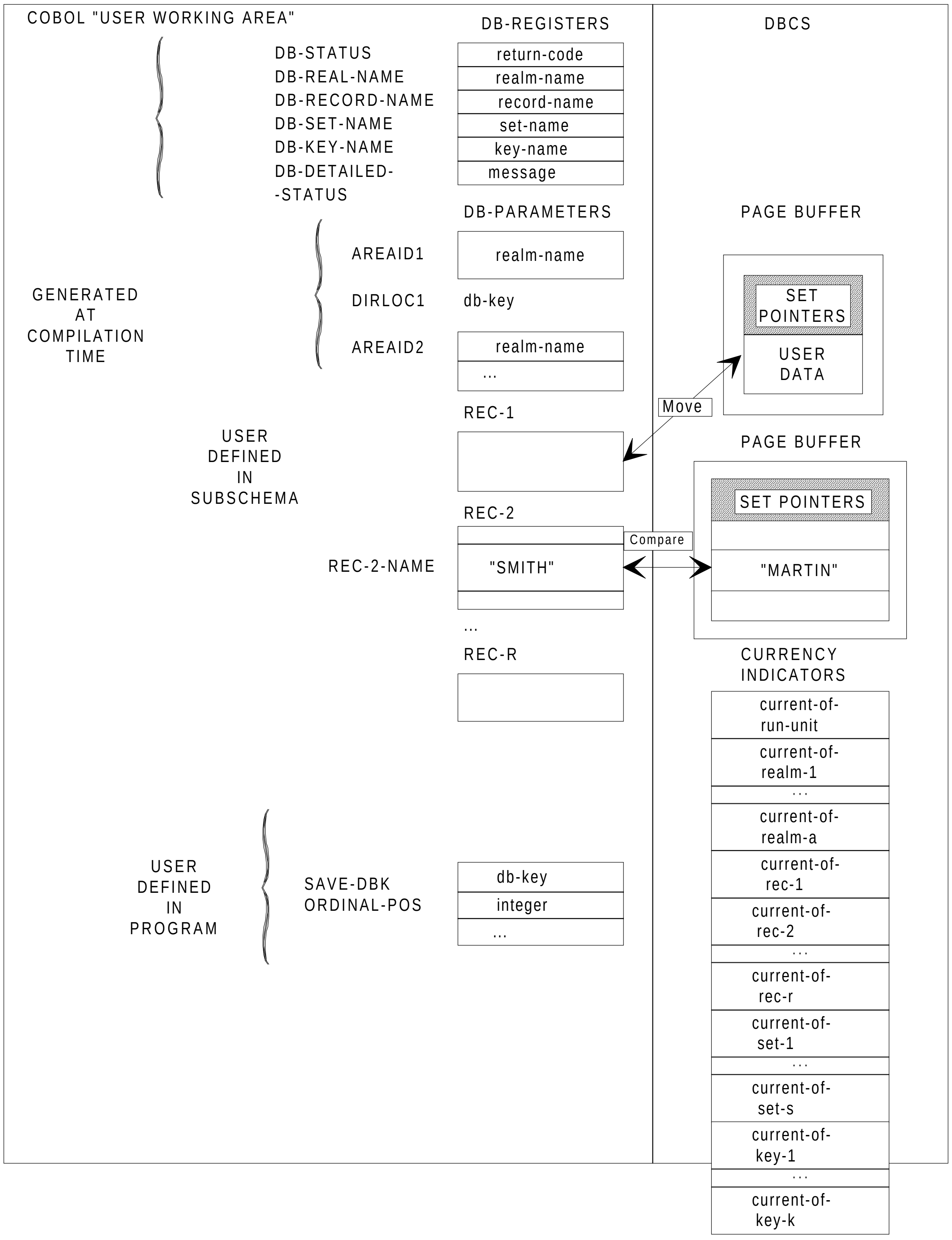


Figure 3-19. Communication between the COBOL program and the DBCS

The programmer is also responsible for all record fields and data-base-parameters required for the following operations:

- the area selection and record location in a STORE
- the area selection in a FIND ANY record-name
- the set selections in a STORE, CONNECT, MODIFY... MEMBERSHIP, and FIND record-name WITHIN set-name [USING identifier].

Conversely, during a "GET record-name" or a "GET identifier ...", the DBCS updates either the entire record or certain fields of that record in the UWA.

Two examples are given below:

Example 1:

Suppose that record REC-1 is to be stored in the database. This record can be located in REALM-3, REALM-4 or REALM-5 depending on the data-base-parameter AREAID1. The record has a LOCATION mode of DIRECT, its area-key being supplied in the data-base-parameter DIRLOC1. Since the record is not member of any set, there is no set selection involved.

The storage sequence might be:

```
  MOVE REC-1-INIT  TO REC-1.  (initialise UWA record)
  MOVE "REALM-5" TO AREAID1.  (select area)
  MOVE 0 TO DIRLOC1.          (suggest first area-key)
  STORE REC-1.                (move the UWA record to
                               the data base)
```

Example 2:

Suppose that record REC-2 is to be retrieved. This record can be located in REALM-1 or REALM-2 depending on the data-base-parameter AREAID2. The record has a LOCATION mode of CALC (with NO DUPLICATES), the CALC-key being the elementary field REC-2-NAME.

The retrieval sequance might be:

```
  MOVE "REALM-1" TO AREAID2.  (select area)
  MOVE "SMITH" TO REC-2-NAME. (initialise Calc-key)
  FIND ANY REC-2.             (locate the record in
                               the data base    )
  GET  REC-2.                 (move the record
                               contents to the UWA)
```

3.6.2 Zones Defined in the UWA by the COBOL Programmer

The zones in the UWA which are defined by the programmer are passed as the input or output parameters of certain DML verbs: "ACCEPT identifier FROM...CURRENCY", "FIND DB-KEY IS identifier", "FIND identifier WITHIN set-name", etc.

Example:

Suppose that the program is interested in the data-base-key assigned to REC-1 by the STORE command in Example 1, above. A field SAVE-DBK with "USAGE IS DB-KEY" is to be defined in the UWA and will be passed as a parameter in the following ACCEPT statement:

```
ACCEPT SAVE-DBK FROM CURRENCY
```

3.6.3 Database Special Registers

Six registers, **DB-STATUS**, **DB-REALM-NAME**, **DB-RECORD-NAME**, **DB-SET-NAME, DB-KEY-NAME** and **DB-DETAILED-STATUS** are automatically generated at compilation time in the UWA. The DBCS uses these registers to communicate information to the COBOL program about the successful or unsuccessful completion of a DML command. The functions of these special registers are explained below.

- DB-STATUS (PIC X(7)) contains a return code. If the command is successful, this code is NULL (0000000). If the command is unsuccessful, the return code is split into a DB-STATEMENT-CODE (PIC XX) containing the code of the command (FIND: "05", GET: "08", etc...) and a DB-STATUS-CODE (PIC X(5)), indicating the cause of the failure (end-of-set or end-of-realm: "02100", record not found: "02400"; etc.).

- After a successful FIND or STORE, DB-RECORD-NAME and DB-REALM-NAME contain the name of the record retrieved (or stored) and the name of its area.

- After an unsuccessful command, DB-RECORD-NAME, DB-REALM-NAME, DB-SET-NAME, or DB-KEY-NAME contain the name of the element relevant to the cause of the failure. These could be:

  - the name of the record
  - and/or the name of the realm
  - and/or the name of the set
  - and/or the name of the key

If no record, realm or set is relevant to the cause of the failure, the corresponding name is filled with blanks.

- After an unsuccessful command, DB-DETAILED-STATUS may contain a variable-length explanatory message complementing the DB-STATUS value. If no message is applicable, the message length is set to 0.

- After an unsuccessful command, the program and the database are left in the state that they were in before the statement was invoked.

DB-STATUS must be tested after each DML command. It is possible to centralize the decoding of DB-STATUS by writing a USE FOR DB-EXCEPTION section in the DECLARATIVES part of the PROCEDURE division.

In the examples throughout this manual, the DB-STATUS check is often omitted for the sake of clarity, in order to draw attention to the DML commands themselves.

3.6.4 Currency Indicators

Currency indicators are entirely maintained by the DBCS and cannot be accessed directly by the programmer. However, from the programmer's point of view, currency indicators are automatic saving zones which are implicitly and constantly referenced in the DML verbs. Currency indicators are fully explained in the subsections on the following pages.

3.7 CURRENCY INDICATORS

3.7.1 Purpose

During the processing of a sequential file, the programmer can issue READ and REWRITE statements without referencing the address of the most recently accessed record to the DBCS. This is possible because the access method maintains an internal **pointer** which contains the address of the record most recently accessed or, in IDS/II terminology, the **current record**. This **currency indicator** (the pointer) is implicitly referenced in data management calls. A REWRITE statement will act upon the **current record**; a READ statement will start from the current record to provide the user with the next record, which becomes in its turn the new **current record**.

In a database, there are many currency indicators. In terms of sets, which can be considered as relatively short sequential files, there is a need for **set** currency indicators, which allow the user to browse through sets by using "FIND NEXT WITHIN set-name" statements. An area can be searched sequentially for records in ascending area-key order by means of "FIND NEXT WITHIN realm-name" statements. This search requires **area** currency indicators.

To meet these needs, the DBCS maintains five categories of currency indicators:

- the **current-of-run-unit**, which is called the **CRU**
- a **current-of-realm** for each realm.
- a **current-of-set-type** for each set-type.
- a **current-of-key-type** for each key-type.
- a **current-of-record-type** for each record-type.

A currency indicator may be in 3 states: present, virtual and null, as described below:

- a **present currency indicator** points to a position (data-base-key) which corresponds to an actual record.
- a **virtual currency indicator** points to a position which does not correspond to an actual record, but to a position between two records.
- a **null currency indicator** does not point to a position.

A virtual currency indicator can only apply to a "current-of-realm" or to a "current-of-set-type" and exists only after an ERASE or DISCONNECT function.

3.7.2 Updating Currency Indicators

Understanding currency indicators is a key to the correct usage of Data Manipulation Language statements (DML). In most cases, the way a DML statement is executed depends on the state of the **currency indicators** before they are invoked by the DML. In addition, following the execution of a DML statement, currency indicators are left in a modified state.

The rules which apply to the updating of currency indicators are specific to each DML verb and will be documented in this manual along with the descriptions of the verbs.

In order to present a general idea of how currency indicators operate, we can describe what happens at the completion of a FIND or STORE statement.

When a record is successfully retrieved (FIND) or stored (STORE), it takes on the folowing statuses:

- the **current-of-run-unit**
- the **current-of-realm** of the area in which it is located
- the **current-of-set-type** of all the set-types in which it is an owner or an actual member
- the **current-of-key-type** of all the key-types on which it is declared in the Schema DDL
- the **current-of-record-type** of its own record-type.

The other currency indicators are left unchanged. These represent records successfully retrieved, stored, or modified previously in the processing.

In certain situations, the automatic updating of the various currency indicators at the completion of a FIND, STORE, MODIFY or CONNECT statement is not desirable. The optional RETAINING clause of these statements, an example of which is shown below, prevents the DBCS from updating the following currency indicators:

- the current-of-realm
- and/or the current-of-record-type
- and/or one or several current-of-set-type(s).

```
                       {  MULTIPLE                        }
                       {  REALM                           }
                       {    RECORD                        }
RETAINING CURRENCY FOR {  || {SETS            }     ||    }
                       {  || { {set-name} . . .}    ||    }
                       {  || {KEYS            }     ||    }
                       {  || { {key-name} . . .}    ||    }
```

The current-of-run-unit, however, is always updated.

3.7.3 Functions of the Currency Indicators

3.7.3.1 Current-of-run-unit

The current-of-run-unit (CRU) is used for sequencing DML verbs. The CRU is an implicit link between the statements (such as FIND and STORE) which establish that a record is the current-of-run-unit and other statements (such as GET, DB Condition (IF), MODIFY, ERASE, CONNECT, DISCONNECT) for which the current-of-run-unit is the **object record**, in other words, the record upon which these statements operate.

3.7.3.2 Current-of-realm

Current-of-realm is used for a sequential search of an area by means of one of the statements below:

```
- FIND NEXT [record-name] WITHIN realm-name.

- FIND PRIOR [record-name] WITHIN realm-name.
```

It can also be used to resume the processing of a record within an area which has temporarily been suspended by the processing of records in other areas. To re-establish the current-of-realm of the area, the following statement can be used:

```
- FIND CURRENT WITHIN realm-name
```

3.7.3.3 Current-of-set-type

Current-of-set-type is used for a sequential search of an occurrence of a set-type by using the statements below:

```
- FIND NEXT [record name] WITHIN set-name.

- FIND PRIOR [record-name] WITHIN set-name.

- FIND DUPLICATE WITHIN set-name USING identifier ....
```

Current-of-set-type is also used to perform a sequential insertion into an occurrence of a set (with the STORE, CONNECT, or MODIFY...MEMBERSHIP statements) when the ORDER is NEXT or PRIOR and the set selection path has only one level defined by APPLICATION.

In addition, current-of-set-type can be used to resume the processing of a record within a set following manipulations in other parts of the database which had no affect on the currency of this set-type. The following statement:

```
- FIND CURRENT WITHIN set-name
```

will re-establish the "current-of-set-type" of this set-type as the "current-of-run-unit".

Current-of-set-type is not only a tool for the identification of a given record in a set occurrence, but it also identifies the set occurrence itself. This is the case when the statements listed below are used:

```
- FIND OWNER WITHIN .set-name

- FIND FIRST [record-name] WITHIN set-.name

- FIND LAST [record-name] WITHIN set-name

- FIND integer [record-name] WITHIN set-name
```

The only connection between the **record** which these statements retrieve and the **current-of-set-type** is that they both participate in the same set occurrence.

The same is true when, during the execution of certain DML verbs, the first level of the **set selection path** has its set occurrence identified by APPLICATION (which means by the current-of-set-type mechanism). The DML for which this applies are listed below:

```
- STORE

- CONNECT

- MODIFY ... set-name MEMBERSHIP

- FIND record-name WITHIN set-name [USING identifier ...]
```

3.7.3.4 Current-of-key-type

Current-of-key-type is used for a sequential search of an occurrence of a key type by means of the following statements:

```
- FIND NEXT [record-name] WITHIN key-name.

- FIND PRIOR [record-name] WITHIN key-name.

- FIND DUPLICATE USING key-name.
```

It can also be used to resume the processing of a record which is connected to a given key following manipulations in other parts of the database which had no affect on the currency of this key-type. The statement:

```
- FIND CURRENT WITHIN key-name.
```

will re-establish the current-of-key-type of this key-type as the **current-of-run-unit.**

3.7.3.5 Current-of-record-type

The current-of-record-type is used only in the statement

```
- FIND CURRENT record-name
```

which re-stablishes the current-of-record-type as the current-of-run-unit.

3.7.4 Example of the Currency Mechanism

Consider the data structure illustrated in Figure 3-20 further on in this subsection, in relation to following problem:

You must search area AR-2 sequentially, and display every C record along with its D members located in area AR-3.

The COBOL sequence might be:

```
LOOP1-INIT. FIND FIRST C WITHIN AR-2.
            GO TO TEST-EOR.
LOOP1.  FIND NEXT C WITHIN AR-2.
TEST-EOR. IF DB-STATUS = "0502100"
          GO TO END-LOOP1.
          GET C.
          DISPLAY ...
LOOP2-INIT. FIND FIRST D WITHIN S.
            GO TO TEST-EOS.
LOOP2.  FIND NEXT D WITHIN S.
TEST-EOS. IF DB-STATUS = "0502100"
          GO TO LOOP1.
          GET D.
          DISPLAY ....
          GO TO LOOP2.
END-LOOP1. ...
```

The currency chart in the second part of Figure 3-20 shows the state of currency indicators following each FIND statement. The effects of the statements in the COBOL sequence above are described in the following paragraphs:

"FIND FIRST C WITHIN AR-2" retrieves the first C record starting from the beginning of the area. After the execution of this statement, C1 takes on the following statuses:

- the current-of-run-unit
- the current-of-realm AR-2
- the current-of-record C
- the current-of-set S
- the current-of-key K.

The other currencies are not modified.

"GET C" moves the contents of the current-of-run-unit to the UWA record, but the currencies are not affected.

"FIND FIRST D WITHIN S" searches the S occurrence S1 identified by the "current-of-set S" (C1) for the first D record, starting from the owner. D3 is retrieved and becomes the following:

- current-of-run-unit
- current-of-realm AR-3
- current-of-record D
- current-of-key J
- current-of-set of S and T.

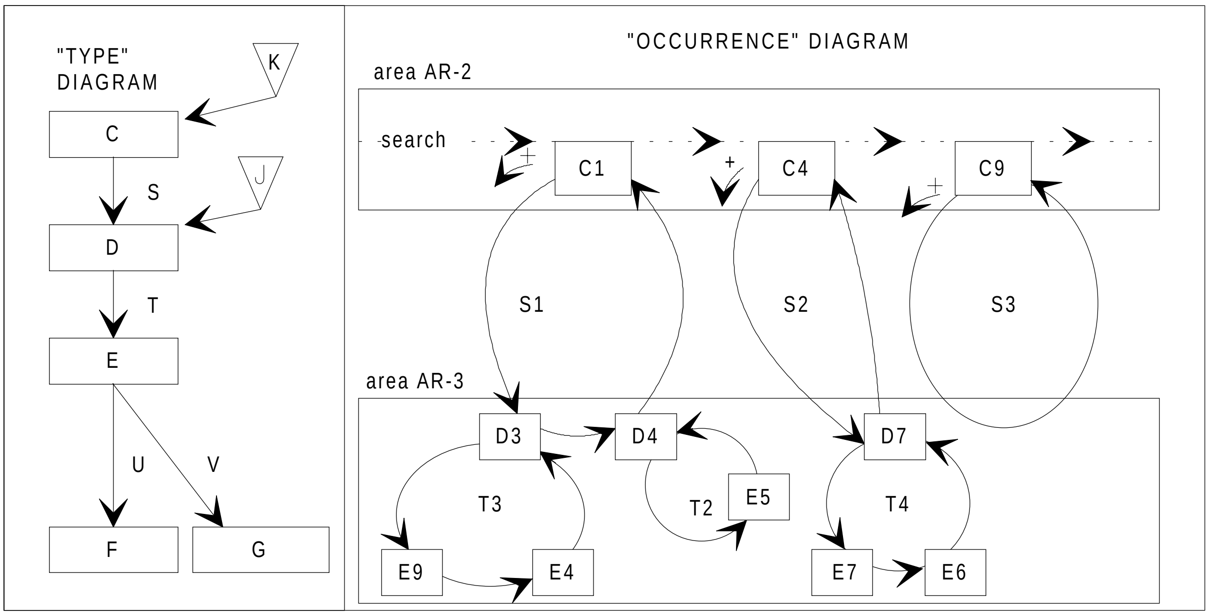
"GET D" does not affect the currencies.

"FIND NEXT D WITHIN S" searches the S occurrence S1 identified by the "current-of-set S" (D3) for the next D record, starting from D3. D4 is retrieved and the processing continues.

When the "end-of-set" condition occurs, the statement is not executed and the currencies are left unchanged.

"FIND NEXT C WITHIN AR-2" starts from the "current-of-realm AR-2" (C1) and retrieves the next C record (C4).

The processing is repeated until the "end-of-realm" condition occurs.



KEY "OCCURRENCE" DIAGRAM

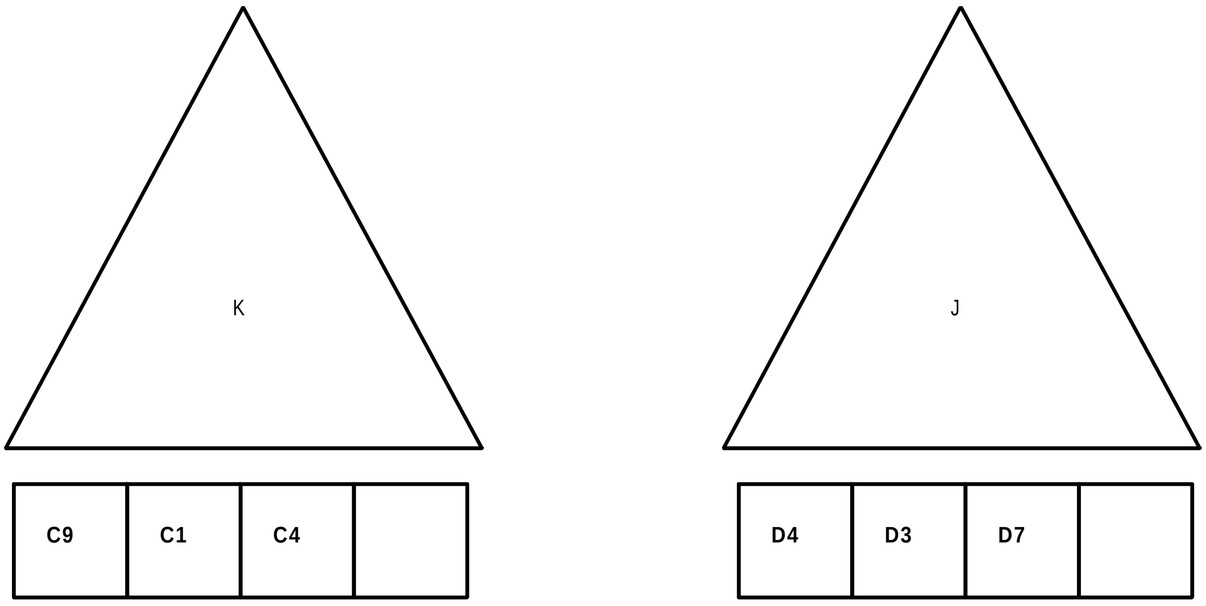


Figure 3-20. The Currency Mechanism (1/2)

CURRENCY CHART

DML STATEMENT	CUR RUN UNIT	CURRENT OF REALM			CURRENT OF RECORD-TYPE			CURRENT OF SET-TYPE			CURRENT OF KEY-TYPE		
		AR-2	AR-3	other	C	D	other	S	T	other	K	J	other
FIND FIRST C WITHIN AR-2	C1	C1	-	-	C1	-	-	C1	-	-	C1	-	-
FIND FIRST D WITHIN S	D3	-	D3	-	-	D3	-	D3	D3	-	-	D3	-
FIND NEXT D WITHIN S	D4	-	D4	-	-	D4	-	D4	D4	-	-	D4	-
FIND NEXT D WITHIN S (EOS)	-	-	-	-	-	-	-	-	-	-	-	-	-
FIND NEXT C WITHIN AR-2	C4	C4	-	-	C4	-	-	C4	-	-	C4	-	-
FIND FIRST D WITHIN S	D7	-	D7	-	-	D7	-	-	D7	-	-	D7	-
FIND NEXT D WITHIN S (EOS)	-	-	-	-	-	-	-	-	-	-	-	-	-
FIND NEXT C WITHIN AR-2	C9	C9	-	-	C9	-	-	C9	-	-	C9	-	-
FIND FIRST D WITHIN S (EOS)	-	-	-	-	-	-	-	-	-	-	-	-	-
FIND NEXT C WITHIN S (EOS)	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 3-20. The Currency Mechanism (2/2)

3.7.5 Example of the Retaining Currency Clause

Now assume that records C and D are located in the same area AR-2 (see Figure 3-21 on the following page).

The preceding sequence must be slightly modified, because each time a D record is retrieved, it becomes the "current-of-realm AR-2" and the statement "FIND NEXT C WITHIN AR-2" will start from D4 (to use an example) instead of C1. If D4 is located between C1 and C4, C4 will be retrieved; if D4 has a lower area-key than C1, C1 will be retrieved again and the program will enter a loop; if D4 has a higher area-key than C4, C4 will be skipped.

To avoid these rather unpredictable results, the RETAINING clause can be added to the two statements labelled LOOP2-INIT and LOOP2, as shown below:

```
LOOP2-INIT. FIND FIRST D WITHIN S RETAINING CURRENCY FOR REALM.
LOOP2.  FIND NEXT D WITHIN S RETAINING CURRENCY FOR REALM.
```

By specifying the RETAINING clause, the retrieval of a D record will not update the "current-of-realm AR-2" and therefore the "FIND NEXT C WITHIN AR-2" will always start from the last C record retrieved.

Note that this solution is not unique. If the statement labelled LOOP1 is broken down as follows:

```
LOOP1. FIND CURRENT C.
       FIND NEXT C WITHIN AR-2.
```

the effect of "FIND CURRENT C" will be to re-establish the "current-of-record C", for example C1, as:

- the "current-of-run-unit"
- the "current-of-realm AR-2"
- the "current-of-key K"
- and the "current-of-set S".

Only the "current-of-realm AR-2" is relevant to the "FIND NEXT C WITHIN AR-2" statement which follows the LOOP1 statement. It ensures that the retrieval will restart from C1.

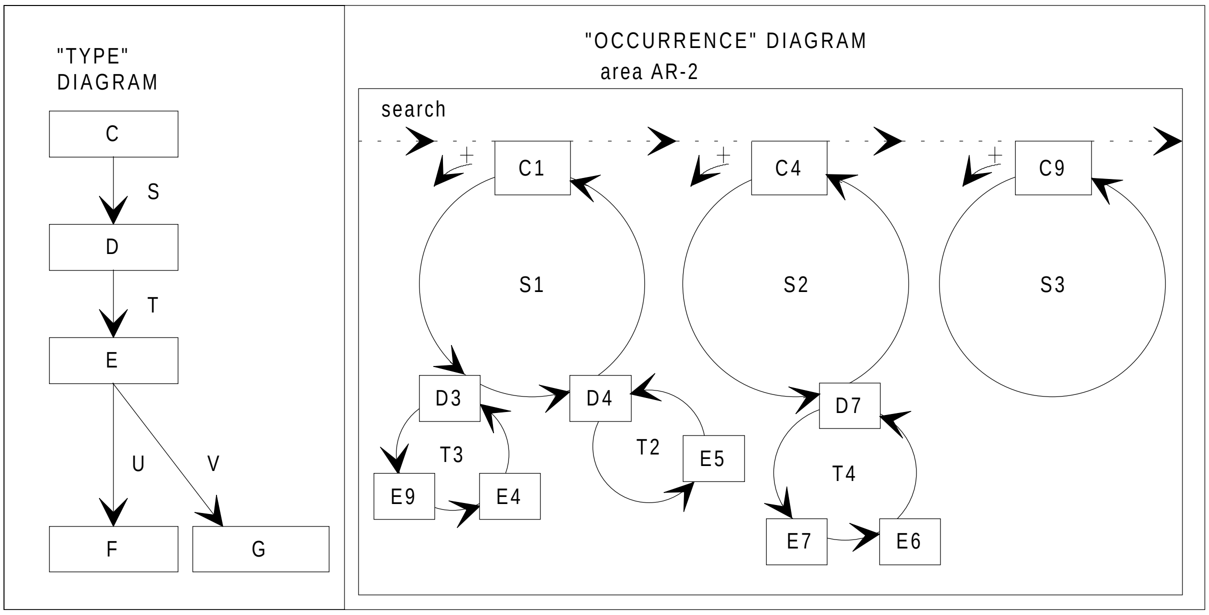


Figure 3-21. "RETAINING CURRENCY" Function (1/2)

CURRENCY CHART

The symbol "-" means that the currency is not modified by the DML statement

DML STATEMENT	CUR RUN UNIT	CURRENT OF REALM		CURRENT OF RECORD-TYPE			CURRENT OF SET-TYPE			CURRENT OF KEY-TYPE		
			other	C	D	other	S	T	other	K	J	other
FIND FIRST C WITHIN AR-2	C1	C2	-	C1	-	-	C1	-	-	C1	-	-
FIND FIRST D WITHIN S RETAINING CURRENCY FOR REALM	D3	-	-	-	D3	-	D3	D3	-	-	D3	-
FIND NEXT D WITHIN S RETAINING CURRENCY FOR REALM	D4-	-	-	-	D4	-	D4	D4	-	I -	D4	-
FIND NEXT D WITHIN S RETAINING CURRENCY FOR REALM (EOS)	-	-	-	-	-	-	-	-	-	-	-	-
FIND NEXT C WITHIN AR-2	C4	C4	-	C4	-	-	C4	-	-	C4	-	-
FIND FIRST D WITHIN S RETAINING CURRENCY FOR REALM	D7	-	-	-	D7	-	D7	D7	-	-	D7	-
FIND NEXT D WITHIN S RETAINING CURRENCY FOR REALM (EOS)	-	-	-	-	-	-	-	-	-	-	-	-
FIND NEXT C WITHIN AR-2	C9	C9	-	C9	-	-	C9	-	-	C9	-	-
FIND FIRST D WITHIN S RETAINING CURRENCY FOR REALM (EOS)	-	-	-	-	-	-	-	-	-	-	-	-
FIND NEXT C WITHIN S (EOS)	-		-	-	-	-	-	-	-	-	-	-

Figure 3-21. "RETAINING CURRENCY" Function (2/2)

3.8 SET MEMBERSHIP CLASS

3.8.1 Purpose

It can be useful to decide at what time a record, which is declared in the schema as member of a set, becomes an actual member of this set.

For example, in the data structure shown in Figure 3-22, the EMPLOYEE record representing a newly-hired employee becomes a member of the set "EMPLOYEES-OF-DEPARTMENT" as soon as that employee joins the company. On the contrary, if the employee spends a period of probation before being assigned to a job, the EMPLOYEE record will be connected to the "JOB-ASSIGNMENT" set only at the end of this period.

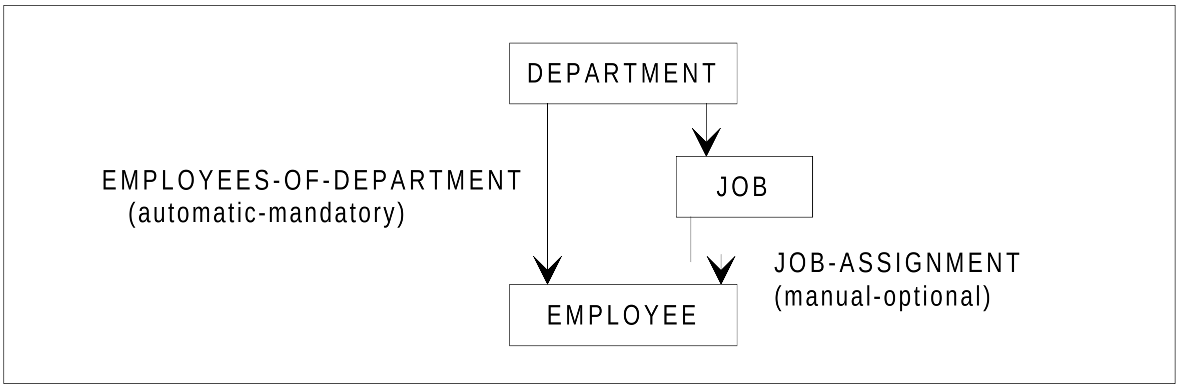


Figure 3-22. AUROMATIC-MANDATORY and MANUAL-OPTIONAL Membership

More generally, the need arises to connect a record to a set on a temporary basis and to disconnect it some time later.

For instance, in a bank system, an account which becomes overdrawn is connected to the "ACCOUNTS-IN-THE-RED" set; when a payment intervenes that re-establishes the credit, the account is disconnected from this set.

3.8.2 Membership Classes

Membership classes are described for each set-type/member-type pair in the MEMBER Subentry of the schema DDL. The related (general) clause is the following:

```
             {AUTOMATIC}              {MANDATORY}
INSERTION IS {         } RETENTION IS {OPTIONAL }
             {MANUAL   }              {FIXED    }
```

AUTOMATIC specifies that the record is made a member of the set when it is stored in the database with a STORE command.

MANUAL specifies that the record is not made a member of the set by the STORE command but only by a later CONNECT command.

MANDATORY specifies that, once connected, the record cannot be removed from the set by a DISCONNECT command. However it may change from one occurrence of the set to a different occurrence of the same set via a MODIFY command.

OPTIONAL specifies that, once connected, the record can be removed from the set by a DISCONNECT command. (It can also change occurences via a MODIFY).

FIXED specifies that, once connected, the record cannot be removed from the set by a DISCONNECT command nor it can change occurences via a MODIFY command.

When **RETENTION is OPTIONAL**, the set is represented on a **type** diagram by a "split" arrow.

All these options are available with IDS/II-V5.

3.8.2.1 Transitions between States

The transitions between the states OUT (not in the database), CONNECTED, and DISCONNECTED are illustrated in Figure 3-23 on the following page.

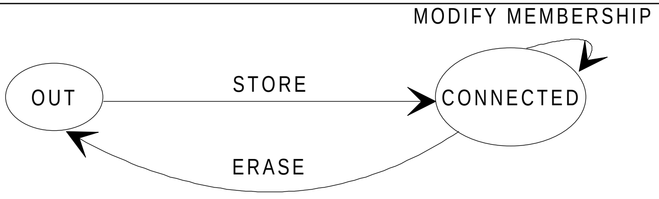
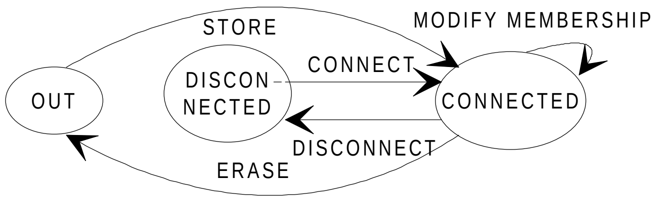
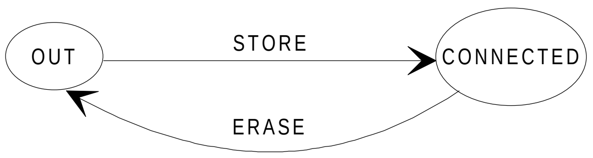
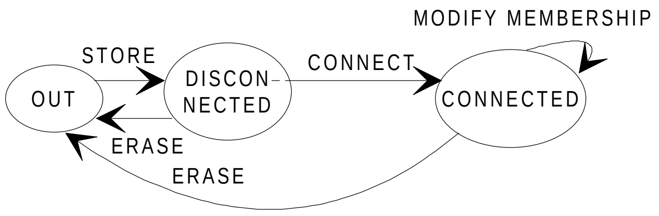
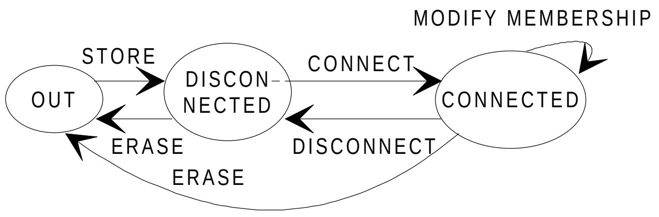
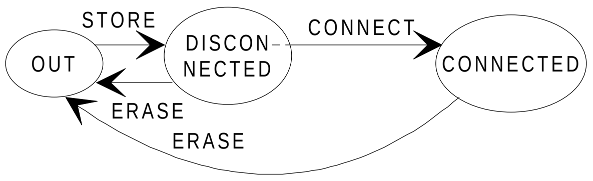
INSERTION	RETENTION	STATE TRANSITIONS
AUTOMATIC	MANDATORY	
AUTOMATIC	OPTIONAL	 STORE MODIFY MEMBERSHIP CONNECT OUT DISCON-NECTED CONNECTED DISCONNECT ERASE
AUTOMATIC	FIXED	
MANUAL	MANDATORY	 MODIFY MEMBERSHIP STORE CONNECT OUT DISCON-NECTED CONNECTED ERASE ERASE
MANUAL	OPTIONAL	 MODIFY MEMBERSHIP STORE CONNECT OUT DISCON-NECTED CONNECTED ERASE DISCONNECT ERASE
MANUAL	FIXED	 STORE CONNECT OUT DISCON-NECTED CONNECTED ERASE ERASE

Figure 3-23. Transitions between States

3.9 SET SELECTION CRITERIA

3.9.1 Purpose

In the IDS/II model, a record has two roles:

- it is a group of fields
- it is a tenant of sets

When a program stores an AUTOMATIC member record in the database, it provides the DBCS with:

- information on its fields (through the UWA record);
- information on the set occurrences to which it is to be connected (through the parameters of the **set selection criteria**).

The set selection criteria are the rules, defined in the DDL, which govern the automatic retrieval of set occurrences by the DBCS. The first role of these criteria is to identify a hierarchy of set-types which forms a continuous path through the data structure leading to the desired set-type. The owner of a set (except for the first set) is a member of the set which is just above it in the hierarchy.

The second role of the set selection criteria is to indicate the **method** of selecting the proper occurrence at each level. The occurrence of the **first** set-type in the hierarchy is selected in one of the following ways:

1) either throught its owner record, if that owner records is an **entry point** record: an entry point record is a record that can be accessed directly because its database-key or CALC-key is accessible in the UWA (LOCATION Mode DIRECT or CALC)

2) or by the current of this set-type (which is equivalent to providing a data-base-key).

After the selection of the **first** set-type, occurrences of other set-types in the hierarchy are selected through their owners. Each owner is identified among the members of the preceding set occurrence in the hierarchy by specifying in the UWA record the contents of some of its elementary fields. The processing stops with the retrieval of the owner of the occurrence of the desired set-type.

In the Schema DDL, the set selection criteria are specified in the SET SELECTION clause of the MEMBER Subentry. The member-types of a multi-member-type set may have different set selection criteria.

The DDL syntax reads:

```
SET SELECTION [FOR set-name-1] IS
 THRU set-name-2 OWNER IDENTIFIED BY
 { APPLICATION                                                  }
 {                                                              }
 { DATA-BASE-KEY [EQUAL TO db-parameter-1]                      }
 {                                                              }
 {          [                         {db-identifier-5}]        }
 { CALC-KEY [db-identifier-4 EQUAL TO {               }] ...    }
 {          [                         {db-parameter-2 }]        }
 {             [AREA-ID EQUAL TO db-parameter-3]                }
 {                [db-identifier-8 EQUAL TO         ]           }
 { KEY key-name-1 [                {db-parameter-5 }] ...       }
 {                [                {db-identifier-9}]           }
 {             [AREA-ID EQUAL TO db-parameter-6  ]              }

[THEN THRU SET-NAME-3 WHERE OWNER IDENTIFIED BY             ]
[ (              [          (db-identifier-7)])             ] . . .
[ (              [          (db-parameter-4 )])             ]
```

3.9.2 Example

Consider the data structure illustrated in Figure 3-24, further on in this section, which represents the staffs of ministries from various countries. All the records are located in the same unique area. The COUNTRY record has a LOCATION mode of CALC with NO DUPLICATES, its CALC-key being the elementary field C-NAME. The PERSON record has a LOCATION mode of VIA STAFF.

Suppose that a new person is appointed to the Staff of the Ministry of Education of FALAND. For this new staff member, a PERSON record **P8** must be stored in the database and inserted into the set occurrence **S4**.

The relevant DDL clauses are given below.

```
  AREA GOVERNMENT.

  RECORD COUNTRY
      LOCATION CALC USING C-NAME DUP NOT
      WITHIN GOVERNMENT.
  02  C-NAME  TYPE CHAR 30.
      ...

  RECORD MINISTRY
      ...
  02  M-NAME  TYPE CHAR 25.

  RECORD PERSON
      LOCATION VIA STAFF
      WITHIN GOVERNMENT.
      ...
  SET ROUND-TABLE
      OWNER COUNTRY
      ...
    MEMBER MINISTRY
      ...
      DUP NOT ALLOWED FOR M-NAME
      ...
  SET STAFF
      OWNER MINISTRY
      ...
    MEMBER PERSON
      ...
```

3.9.2.1 Two-level Set Selection Path

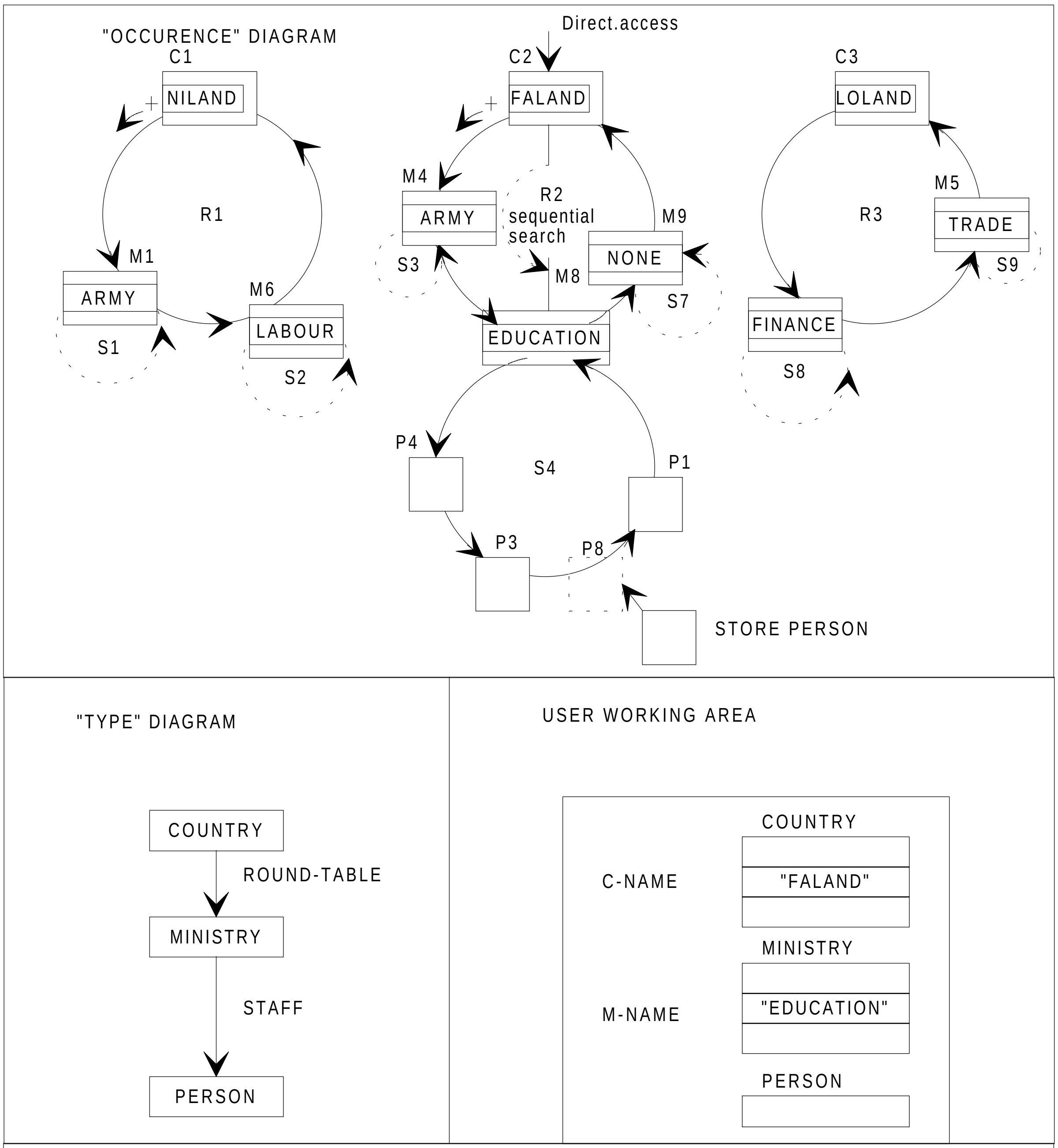
The set selection clause of PERSON as member of STAFF can be defined as follows:

```
SET SELECTION IS
     THRU ROUND-TABLE OWNER IDENTIFIED BY CALC-KEY
     THEN THRU STAFF WHERE OWNER IDENTIFIED BY M-NAME.
```

The storage sequence of P8 could be:

```
 MOVE "FALAND" TO C-NAME.
 MOVE "EDUCATION" TO M-NAME.
 STORE PERSON.
```

The DBCS will use C-NAME to identify C2, thus R2, then search R2 for a MINISTRY record whose M-NAME field contains "EDUCATION". M8 is located and serves to identify S4. Note that the set selection is functionally used twice: first to assign a data-base-key to P8 (LOCATION VIA STAFF), secondly to insert P8 logically into STAFF.



```
DDI.     SET NAME IS STAFF
           ...
         MEMBER IS PERSON
           ...
          SET SELECTION FOR STAFF IS

            THRU ROUND-TABLE OWNER IDENTIFIED BY CALC-KEY
            THEN THRU STAFF WHERE OWNER IDENTIFIED BY C-NAME
```

Figure 3-24. Set Selection Criteria

3.9.2.2 One-level Set Selection Path

Suppose now that there is a major change in the ministerial staff which results in new personnel being appointed in groups. In that case it would be useless **from a performance point of view**, to go through the identification of C2 and M8 for each new PERSON record. The "set selection" clause of PERSON can then be written:

```
 SET SELECTION IS
     THRU STAFF WHERE OWNER IDENTIFIED BY APPLICATION.
```

This means that the program must first initialize the "current-of-set STAFF" and then store the new PERSON records in a sequence.

The storage sequence might be:

```
 MOVE "FALAND" TO C-NAME.     Locate C2 which becomes
 FIND ANY COUNTRY.            "current-of-set ROUND-TABLE".
 MOVE "EDUCATION" TO M-NAME.  within R2,locate M8
 FIND MINISTRY WITHIN         wich becomes :
      ROUND-TABLE             "current-of-set STAFF"
      CURRENT USING M-NAME.   and identifies S4.

LOOP. READ NEWLY-APPOINTED-FILE AT END GO TO END-LOOP.
      MOVE NEWLY-APPOINTED-RECORD TO PERSON.
      STORE PERSON.
      GO TO LOOP.             PERSON becomes the new
                              "current-of-set STAFF"
                              but it still identifies S4
                              for the next STORE.
END-LOOP....
```

In conclusion, if performance is a major concern, the set selection clause should be adapted to the way member records are chronologically entered into the database.

3.9.3 Uniqueness of the Selected Set Occurrence

The set selection criteria must uniquely identify the final set occurrence. There must be no ambiguity in the choice of set occurrences at each level in the hierarchy.

3.9.3.1 First Level of the Hierarchy

The condition of uniqueness is met automatically in these two ways:

- if the owner has a LOCATION mode of DIRECT and is selected by its data-base-key (OWNER IDENTIFIED BY DATA-BASE-KEY), or
- if the owner is retrieved from the "current-of-set" of this set-type (OWNER IDENTIFIED BY APPLICATION)

In both cases, the value supplied is a data-base-key which is unique within the database.

When the owner has a LOCATION mode of CALC and is selected by its CALC-key, the condition of uniqueness is satisfied only in the following two cases:

1) if the area containing the record is specified in the AREA-ID parameter when a choice exists between different areas. The transfomation from CALC-key to area-key uses parameters such as the bucket size in pages or the number of buckets, which are specific to each area. It is a localized transformation.

2) if the LOCATION clause includes a "DUPLICATES ARE NOT ALLOWED" phrase which prevents the storage in the same area of more than one record of this type having a given CALC-key value. Like the area-key computation mentioned above, a "NO DUPLICATES" check does cover the entire database. It is limited to the records of the area selected.

To statement below summarizes the uniqueness condition:

```
 CALC-KEY [ + area-name ] + "DUPLICATES NOT"
                ========>   unique identification
```

When the owner is selected by a secondary key, the selection condition is satisfied only if both conditions below are fulfilled:

- the area containing the record is specified in the AREA-ID parameter, if the range of the key is not the entire database
- the secondary key clause includes a "DUPLICATES ARE NOT ALLOWED" phrase.

```
SECONDARY KEY [ + area-name ] + "DUPLICATES NOT"
                 ========>   unique identification
```

3.9.3.2 Second Level of the Hierarchy and Further Levels

The condition of uniqueness is met if the **elementary fields** that identify their owner as a member record of the preceding set in the hierarchy are either a) or b) below:

a) these elementary fields are the object of a "DUPLICATES ARE NOT ALLOWED FOR..." clause in the MEMBER Subentry of the corresponding set, as illustrtated in the following DDL statements:

```
 SET NAME IS ROUND-TABLE
     ...
 MEMBER IS MINISTRY
     ...
     DUPLICATES ARE NOT ALLOWED FOR M-NAME
     ...
```

b) these elementary fields are the sort-keys defined in the KEY Clause of the corresponding set, with a "DUPLICATES ARE NOT ALLOWED" phrase in the ORDER clause or KEY clause. In such a situation, if set ROUND-TABLE were sorted on M-NAME, the related DDL clauses would be:

```
SET NAME IS ROUND-TABLE
     ...
     ORDER IS PERMANENT INSERTION IS
      SORTED BY DEFINED KEYS
       DUPLICATES ARE NOT ALLOWED.
     ...
 MEMBER IS MINISTRY
     ...
     KEY IS ASCENDING M-NAME
     ...
```

NOTE: If a CALC-key is used as an identifier at a level other than the first level of the hierarchy, it is submitted to the rules concerning DUPLICATES which are listed in the subsection "First Level of the Hierarchy", earlier in this section. In effect, "DUPLICATES NOT" in the LOCATION mode, whose scope is the **area** and not the **database**, does not guarantee the uniqueness of the CALC-key value within a set occurrence which spans several areas.

3.9.4 Multiple of Set Selection Paths

When a STORE command is applied to a record, the DBCS follows as many set selection paths as there are sets in which the record is declared as an automatic member. If at least two occurrences of the same record-type are implied in different paths and if the same identifiers are used for their identification, the problem arises of supplying the values of these identifiers for each occurrence.

The solution is supplied by the "EQUAL TO" phrases of the "SET SELECTION" clause.

- At the first level of the hierarchy,

```
DATA-BASE-KEY EQUAL TO db-parameter-1
```

specifies that the value of the data-base-key of the record is to be found in "db-parameter-1" and not in the parameter defined in the "LOCATION DIRECT db-parameter" clause. Also at the first level,

```
         {                          {db-identifier-2}
CALC-KEY { db-identifier-1 EQUAL TO {db-parameter-1 } ...
         {                                          }

         AREA-ID EQUAL TO db-parameter-2
```

specifies that the value of the CALC-key of the record is to be found in "db-identifier-2" or "db-parameter-1" and not in the CALC-key fields "db-identifer-1" of the UWA record. Secondly, the area-name identifying the area of a multi-area CALC record is to be found in db-parameter-2 and not in the AREA-ID parameter defined in the WITHIN Clause of this record.

"APPLICATION" cannot be redefined since there is only one "current-of-set" per set-type.

- At the second of the hierarchy and further levels,

```
                         {db-identifier-2}
db-identifier-1 EQUAL TO {               }
                         {db-parameter-1 }
```

specifies that the value of every field identifying the record is to be found in "db-identifier-2" or "db-parameter-1" and not in the field "db-identifier-1" of the UWA record.

Whatever the level of hierarchy, the db-identifiers to the right of "EQUAL TO" must have exactly the same TYPE characteristics as those of the db-identifiers to the left of EQUAL TO.

3.9.4.1 Example 1. Redefinition of the CALC-key in the UWA

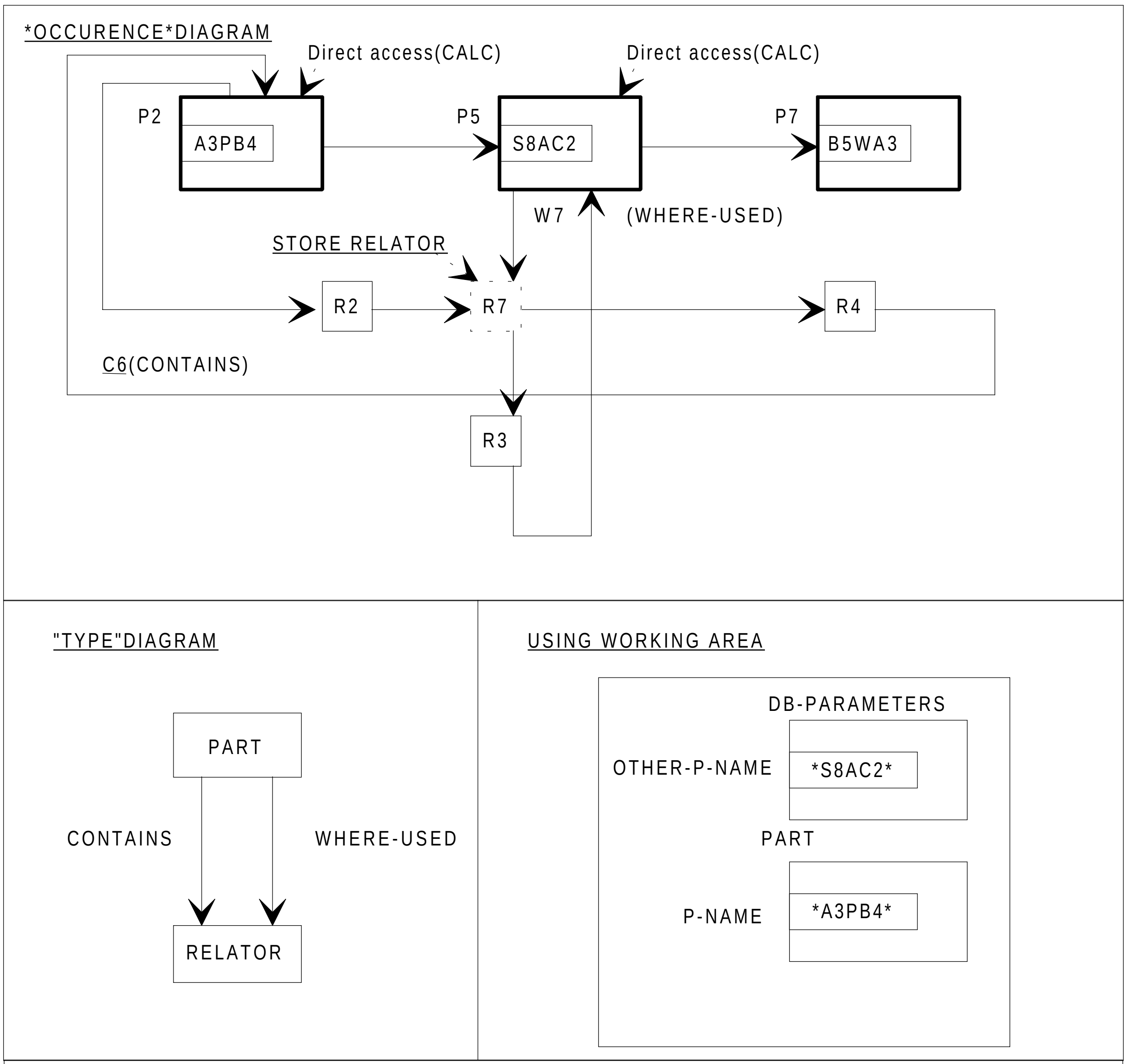
Consider the "parts-explosion" data structure shown in Figure 3-25, further on. Assume, temporarily, the following:

- that all the records are located in only one area
- that record PART has a LOCATION mode of CALC with DUPLICATES NOT, the CALC-key being P-NAME.
- and that record RELATOR has a LOCATION mode of VIA CONTAINS.

The relevant DDL clauses are shown below:

```
AREA   AR-1.
RECORD PART
       LOCATION CALC USING P-NAME DUP NOT
       WITHIN AR-1.
02     P-NAME TYPE CHAR 5.
       ...
RECORD LOCATOR
       LOCATION VIA CONTAINS
       WITHIN AR-1.
       ...
SET    CONTAINS
       OWNER PART
       ...
MEMBER RELATOR
       ...
SET    WHERE-USED
       OWNER PART
       ...
MEMBER RELATOR
       ...
```

When a RELATOR record is stored, the set selection paths of CONTAINS and WHERE-USED imply two occurrences of the PART record-type.



```
DDL       SET NAME IS CONTAINS
              ...
          MEMBER IS RELATOR
              ...
           SET SELECTION FOR CONTAINS IS
              THRU CONTAINS OWNER IDENTIFIED BY CALC-KEY
          ...
          SET NAME IS WHERE-USED
              ...
          MEMBER IS RELATOR
              ...
            SET SELECTION FOR WHERE-USED IS
               THRU WHERE-USED OWNER IDENTIFIED BY
                  CALC-KEY P-NAME EQUAL TO OTHER P-NAME
```

Figure 3-25. Multiple Set Selection Paths

They can be defined as follows:

```
SET SELECTION FOR CONTAINS IS
```

```
    THRU CONTAINS OWNER IDENTIFIED BY CALC-KEY
...
SET SELECTION FOR WHERE-USED IS
    THRU WHERE-USED OWNER IDENTIFIED BY
    CALC-KEY P-NAME EQUAL TO OTHER-P-NAME.
```

The storage sequence of record R7 is then:

```
MOVE "A3PB4" TO P-NAME.
MOVE "S8AC2" TO OTHER-P-NAME.
STORE RELATOR.
```

P2, the owner of the CONTAINS occurrence **C6**, is identified by the value "A3PB4" of the "CALC-key" supplied in field P-NAME of the UWA PART record. **P5**, the owner of the WHERE-USED occurrence **W7**, is identified by the value "S8AC2" of the CALC-key supplied in db-parameter OTHER-P-NAME.

3.9.4.2 Example 2. Redefinition of the CALC-key and AREA-ID in the UWA

Suppose now that the PART records are located in two areas, AR-1 and AR-2, with the data-base-parameter AR-ID, and that the RELATOR records are located "WITHIN AREA OF OWNER" (of set CONTAINS).

The DDL AREA and RECORD Clauses become:

```
AREA   AR-1.
AREA   AR-2.
RECORD PART

       LOCATION CALC USING P-NAME DUP NOT
       WITHIN AR-1 AR-2 AREA-ID IS AR-ID
02     P-NAME TYPE CHAR 5.
       ...
RECORD LOCATOR
       LOCATION VIA CONTAINS
       WITHIN AREA OF OWNER.
      ...
```

The Set Selection Clause of WHERE-USED must be further extended to allow for the redefinition of the AREA-ID parameter,as shown on the following page.

```
SET SELECTION FOR WHERE-USED IS
    THRU WHERE-USED OWNER IDENTIFIED BY
    CALC-KEY P-NAME EQUAL TO OTHER-P-NAME
            AREA-ID EQUAL TO OTHER-AR-ID.
```

If P2 is located in AR-1 and P5 in AR-2, the storage sequence of record R7 becomes:

```
MOVE "A3PB4" TO P-NAME.
MOVE "AR-1" TO AR-ID.
MOVE "S8AC2" TO OTHER-P-NAME.
MOVE "AR-2" TO OTHER-AR-ID.
STORE RELATOR.
```

3.9.4.3 Activating of the Set Selection Mechanism

So far in this discussion, set selection has been associated with the insertion of a record into a set during a STORE command.

Theoretically, set selection can be considered as a **procedure** which is implicitly invoked by the DBCS when processing certain DML commands. The function of the Set Selection Clause of the schema is to define the parameters of such a procedure.

Since it is the responsibility of the programmer to initialize these parameters prior to a DML call, it is essential to know which DML commands are likely to activate the set selection and in what circumstances. The exhaustive list of conditions for invoking the set selection procedure is given below.

1) The STORE command triggers set selection mechanisms for the following sets:

   for the set specified in the VIA phrase when the record has a LOCATION mode of "VIA set-name" (area-key assignment). This occurs whether the record is an AUTOMATIC or MANUAL member of that set;

   for all sets of which the record is an AUTOMATIC member (insertion).

2) The CONNECT command triggers the set selection mechanism:

   for the set specified in the command

3) The MODIFY command triggers the set selection mechanism:

   for all sets specified in the MEMBERSHIP phrase (insertion into a different occurrence). This occurs whether the record is a MANDATORY member or an OPTIONAL member currently connected.

4) The FIND command triggers the set selection mechanism:

   for the set specified in the format below, which accesses a record via a set sellection path:

   ```
   FIND record-name WITHIN set-name [USING identifier ...].
   ```

When the set selection mechanism is not limited to one level identified by APPLICATION, it can be considered as an automatic structural chek, since it is the DBCS and not the programmer that checks the existence of all owners down through the hierarchy. If access rights were available at record level, a program could be allowed to connect a record without having the right to access the owners of the set selection path.

3.10 SET ORDERING CRITERIA

3.10.1 Purpose

Each time a member record is to be connected to a set, the DBCS must determine the following:

- the set occurrence where the record is to be inserted
- the logical position of the record within this set occurrence relative to the other members which are already connected.

The set occurrence is selected by the **set selection criteria.**

The relative position of the record within the set occurrence is determined by the **set ordering criteria** defined in the ORDER Clause of the SET Entry. These criteria are based on:

- the chronological order of insertion (INSERTION FIRST, LAST)
- the position of the "current-of-set-type" (INSERTION NEXT, PRIOR)
- the contents of specified fields of the record (INSERTION SORTED).

In a multi-member-type set, the insertion order is either:

a) the same for any record-type if INSERTION IS FIRST, LAST, NEXT, PRIOR or SORTED BY DEFINED KEYS. In the latter case, keys with identical TYPE characteristics must be defined in every record-type of the set.

b) or, these functions depend on the record-type if INSERTION IS SORTED WITHIN RECORD-TYPE. If this is the case, the sort-keys are defined independently for each record-type. When a record is inserted, its position is determined only with respect to the other records of its type already in the set occurrence. Records of a different record-type are ignored.

The following subsections give a complete description of all the possible methods of record insertion. Each description is accompanied by the DDL syntax of the ORDER Clause of the SET Entry.

3.10.2 Order FIRST

The DDL syntax which specifies that a record be inserted first is given below:

```
ORDER IS PERMANENT INSERTION IS FIRST
```

The record is inserted before all the members which already present in the set, if any. In this instance, **before** refers to the conventional order chosen for the set occurrence and represented in the occurrence diagrams by the symbol "+" (See Figure 3-26).
If the members are retrieved later by "FIND NEXT WITHIN set-name" statements, they will appear in the reverse-chronological order of insertion (FIRST IN LAST OUT).

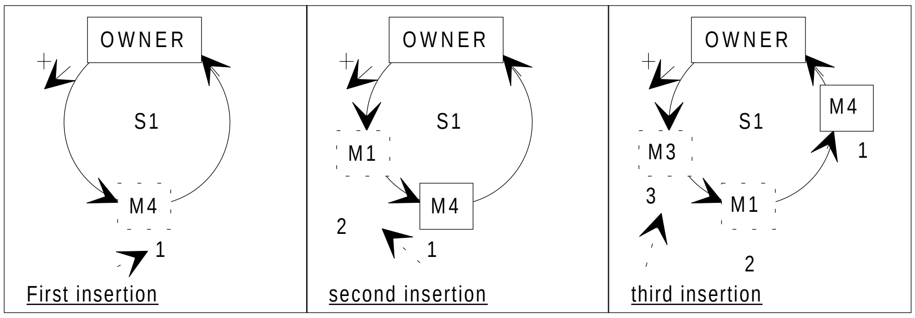


Figure 3-26. Order FIRST

3.10.3 Order LAST

The DDL syntax which specifies that a record be inserted last is given below:

```
ORDER IS PERMANENT INSERTION IS LAST
```

The record is inserted after all the records already present, if any. In this context, **after** refers to the conventional set occurrence order (see Figure 3-27). If the members are later retrieved by "FIND NEXT WITHIN set-name" statements, they will appear in the chronological order of insertion (FIRST IN FIRST OUT).

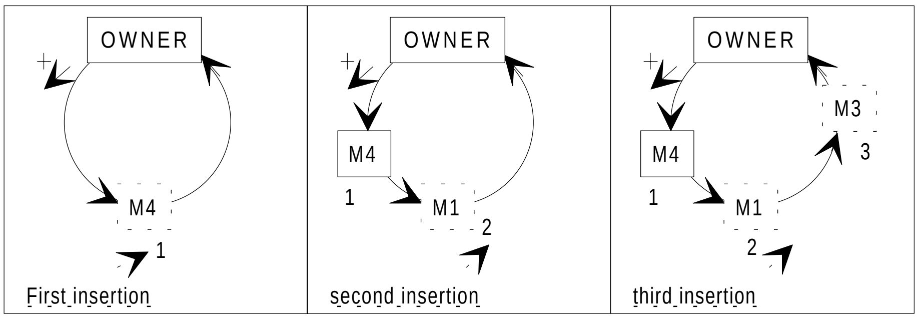


Figure 3-27. Order LAST

3.10.4 Order NEXT

The DDL syntax which specifies that a record be inserted next is given below:

```
        ORDER IS PERMANENT INSERTION IS NEXT
```

The record is inserted just after the **current-of-set-type**, according to the conventional set occurrence order. Figure 3-28 illustrates the ordering method. The current-of-set-type is chosen arbitrarily in each case; it is not the result of the preceding insertion.

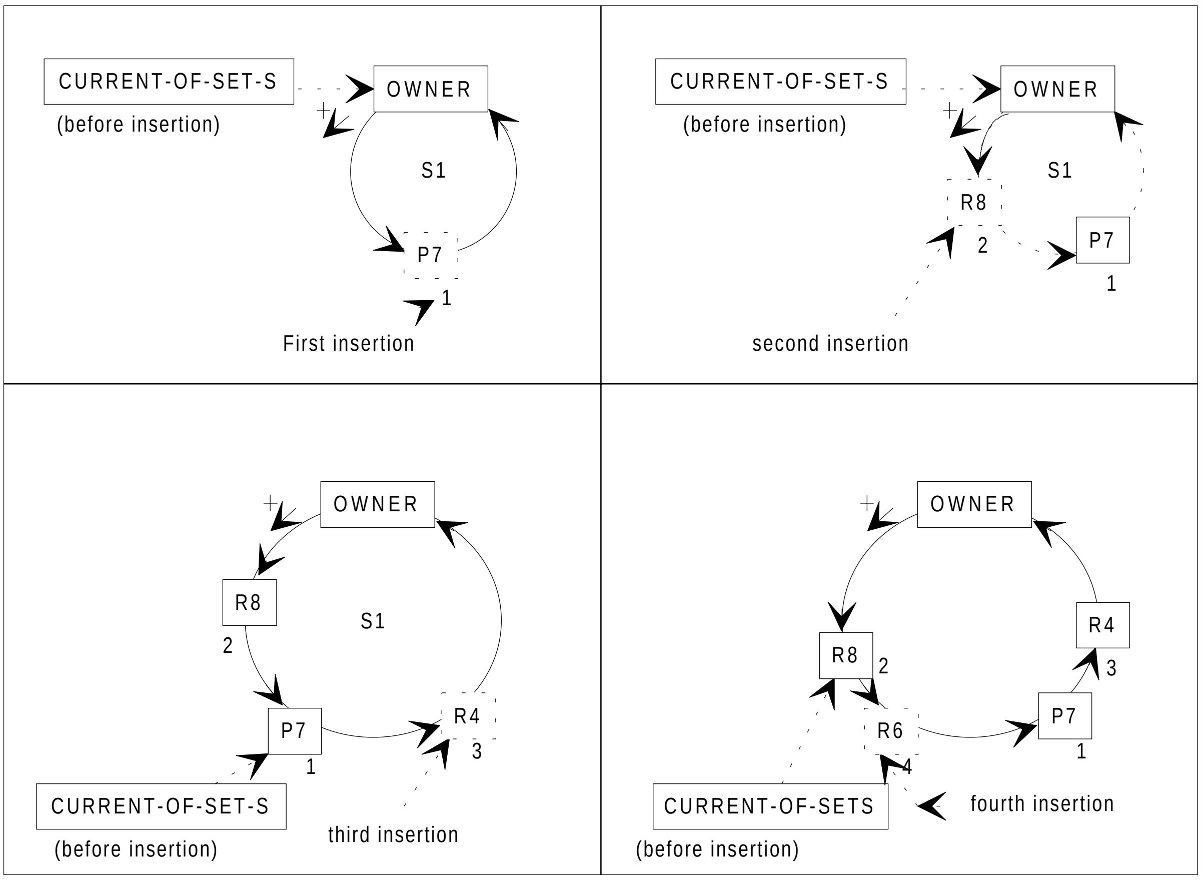


Figure 3-28. Order NEXT

The ORDER Clause may be contradictory to the SET SELECTION Clause in the following situations:

- If the set occurrence is selected by a set selection path composed of only one level identified by APPLICATION (by the current-of-set-type), then the insertion is performed with respect to this **current-of-set-type**, as described above.
- If all the members are inserted in a row, each record inserted becoming the current-of-set-type for the following insertion, the ORDER NEXT Clause has the same final effect as the ORDER LAST Clause.
- If the set occurrence is selected by any other type of set selection path, then the final occurrence is identified by its owner, which is considered as an intermediate current-

of-set-type. The insertion is performed with respect to this owner, which makes ORDER NEXT equivalent to ORDER FIRST.

NOTE: The actual current-of-set-type may point to a record participating in the selected set occurrence or to a record in a different set occurrence; it is ignored by the DBCS (see Figure 3-29).

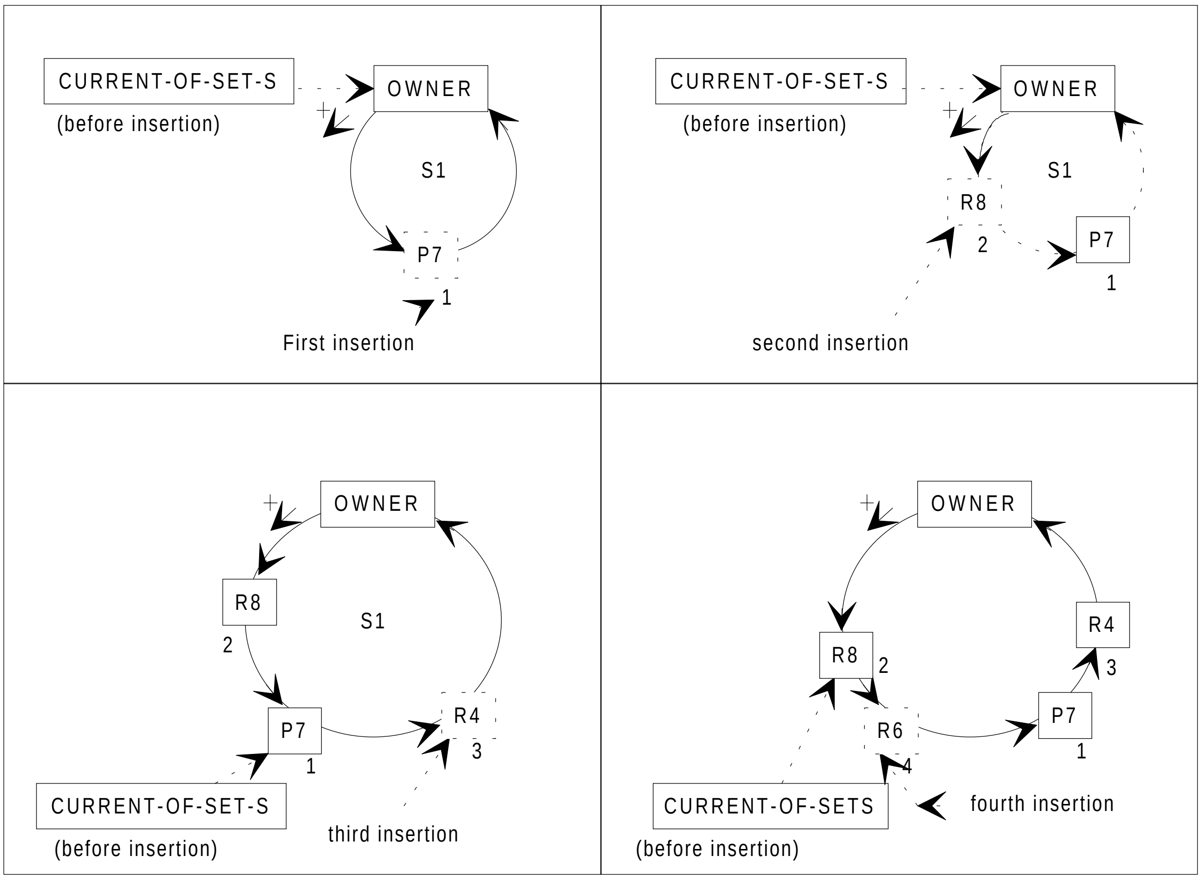


Figure 3-29. OrderNEXT When Set Selection is Not By APPLICATION

In conclusion, when an order of NEXT or PRIOR is functionally required by the program logic, a set selection path should be defined which is composed of **only one level** identified by APPLICATION.

3.10.5 Order PRIOR

The DDL syntax which specifies that a record is to be inserted **prior**, is given below.

```
ORDER IS PERMANENT INSERTION IS PRIOR
```

The record is inserted just before the current-of-set-type, according to the conventional set occurrence order.

Figure 3-30 illustrates this ordering method. The current-of-set-type is chosen arbitrarily in each case and is not the result of the preceding insertion.

The possible conflict between ORDER NEXT and the set selection holds true for ORDER PRIOR; it may become equivalent to ORDER LAST if the set selection path has more than one level identified by APPLICATION.

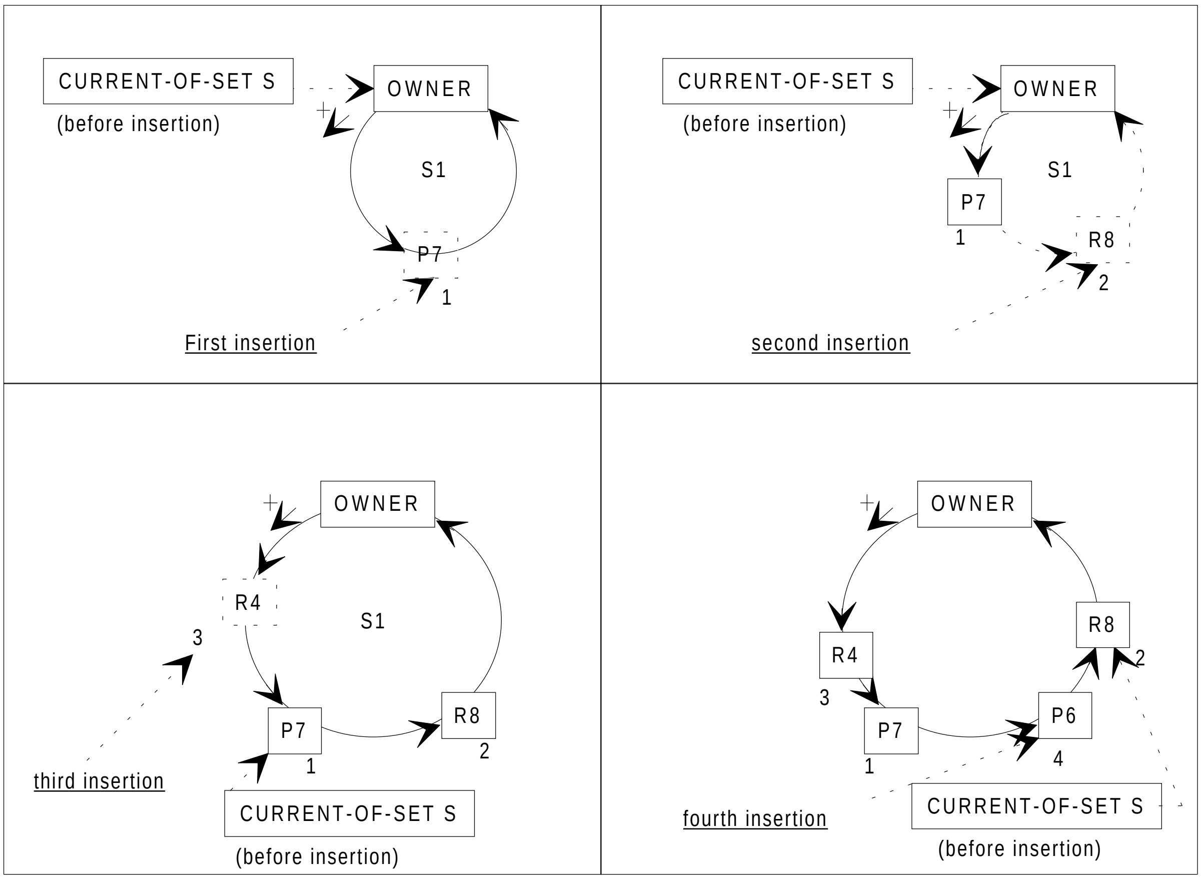


Figure 3-30. Order PRIOR

3.10.6 Order SYSTEM-DEFAULT

The DDL syntax that specifies that a record is to be inserted by system default is given below :

```
ORDER IS PERMANENT INSERTION IS SYSTEM-DEFAULT
```

The record is inserted in the order most convenient for the DBCS.

3.10.7 Order SORTED BY DEFINED KEYS (mono-member-type set)

The DDL syntax which specifies that a record is to be inserted by defined key is given below:

```
SET NAME IS . . .

    OWNER IS . . .

    ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS

                                {FIRST              }
              DUPLICATES ARE {LAST               }
                                {NOT ALLOWED        }
                                {SYSTEM-DEFAULT     }

MEMBER IS . . .
          . . .

[        {ASCENDING } {data-base-identifier-2}        ]
[ KEY IS {          } {RECORD-TYPE           }        ]
[        {DESCENDING} {DATA-BASE-KEY         }        ]
[                                                     ]
[      [[ ASCENDING ] {data-base-identifier-3}]       ]
[      [[           ] {RECORD-TYPE           }] . . . ]
[      [[ DESCENDING] {DATA-BASE-KEY         }]       ] . . .
```

The position of the record is determined by comparing its sort-key with that of the members already present in the set, if any.

The key is made up of one or several key items. These items may be elementary fields of the record or its data-base-key. Key items are stated in the order of priority, the global comparison proceeding from left to right. A key item, except for the first, is therefore involved in the processing only if the comparison of the key items to its left has produced a match. The rules of comparison are adapted to the TYPE of the key item.

The following sequences of reference hold true:

- the EBCDIC collating sequence for the CHARACTER type
- the algebraic order for the DECIMAL and BINARY types
- the arithmetic order for the DATA-BASE-KEY.

The meaning of the parameters in the DDL are given below:

ASCENDING means that, according to the sequence of reference for the item type, if the value of a key item of record A1 is greater than the corresponding key item of record A2, then record A1 must be inserted after record A2, according to the conventional set order.

DESCENDING means that, according to the sequence of reference of the item type, if the value of a key item of record A1 is greater than the corresponding key item of record A2, then record A1 must be inserted before record A2, according to the conventional set order.

The keywords ASCENDING and DESCENDING need not be repeated if they are the same for other key items which follow in the same statement, for example:

```
DESCENDING YEAR MONTH ASCENDING NUMBER FULL-NAME
```

is equivalent to

```
DESCENDING YEAR DESCENDING MONTH ASCENDING NUMBER ASCENDING FULL-NAME
```

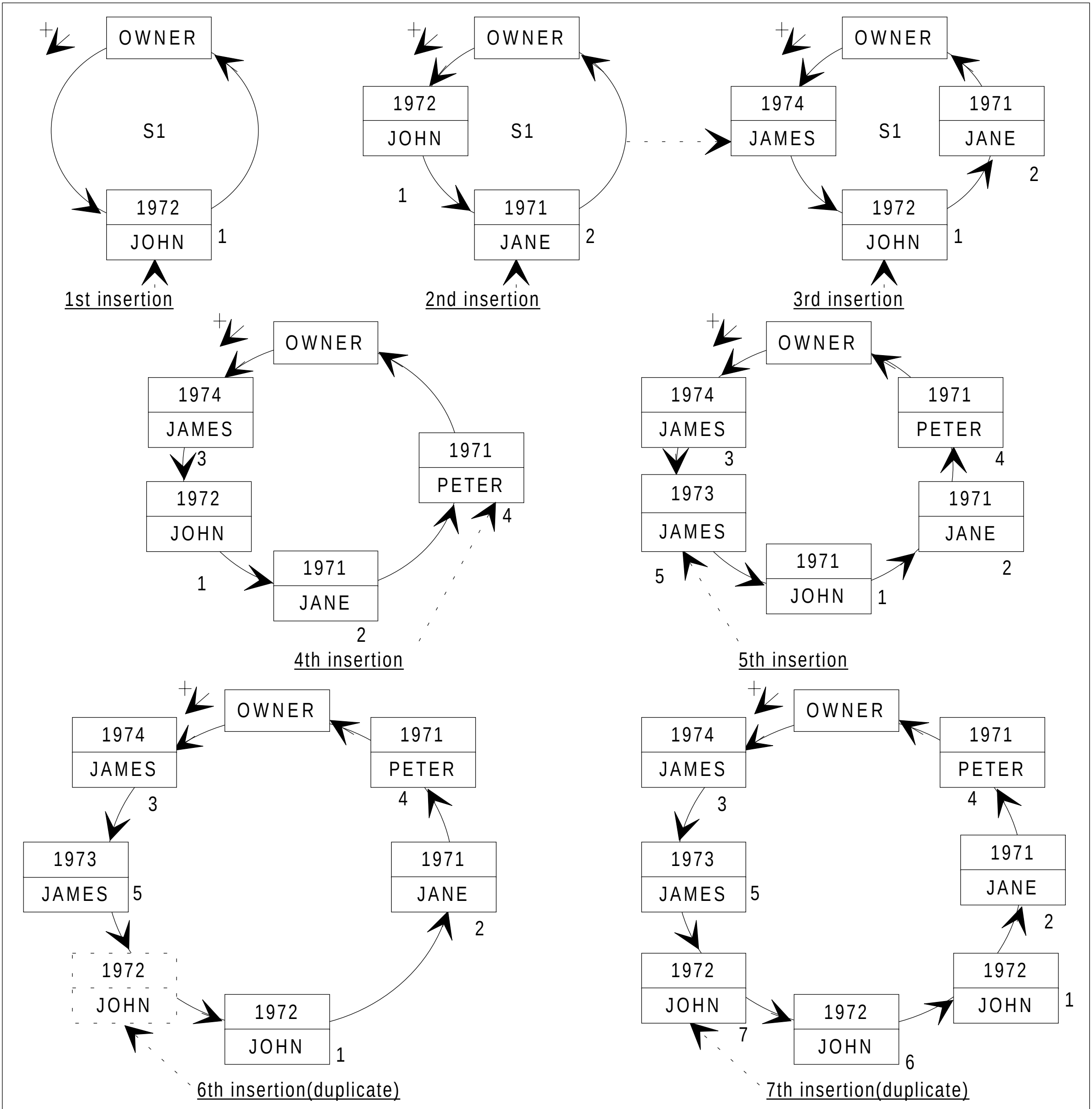
DATA-BASE-KEY may be specified only once in the KEY Clause and as the last key item: since two data-base-keys cannot be equal, the key items to the right would never be involved in the comparison.

When the record to be inserted has the same sort-key value as that of a member of the set occurrence, it is dealt with according to the specifications in the DUPLICATES phrase:

- If DUPLICATES ARE NOT ALLOWED, the insertion is denied and a data-base-exception occurs.
- If DUPLICATES ARE FIRST, the record is inserted just ahead of the group of duplicates.
- If DUPLICATES ARE LAST, the record is inserted just after the group of duplicates.

The "DUPLICATES NOT" phrase must be specified if DATA-BASE-KEY is a key item.

Figure 3-31 illustrates the ordering method:



```
SCHEMA DDL RELEVANT CLAUSES
RECORD NAME IS M
   ...
  02 BIRTH-YEAR TYPE IS SIGNED UNPACKED DECIMAL 4.
  02 FIRST-NAME TYPE IS CHARACTER 30.
   ...
SET NAME IS S
   ...
  ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS
                                  DUPLICATES ARE FIRST.
MEMBER IS M
   ...
  KEY IS DESCENDING BIRTH-YEAR ASCENDING FIRST-NAME
```

Figure 3-31. Order SORTED BY DEFINED KEYS (Mono-member-type Set)

3.10.8 Order SORTED BY DEFINED KEYS (multi-member-type set)

The DDL syntax for a record to be inserted by defined keys in multi-member-type set is given below:

```
SET NAME IS . . .

    OWNER IS . . .

    ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS

        [ RECORD-TYPE SEQUENCE IS { record-name } ...]

                                   {FIRST             }
                  DUPLICATES ARE {LAST              }
                                   {NOT ALLOWED       }
                                   {SYSTEM-DEFAULT    }

(and for each member)

MEMBER IS . . .
          . . .
[        {ASCENDING } {data-base-identifier-2}     ]
[ KEY IS {          } {RECORD-TYPE           }     ]
[        {DESCENDING} {DATA-BASE-KEY         }     ]
[                                                  ]
[      [[ ASCENDING ] {data-base-identifier-3}]    ]
[      [[           ] {RECORD-TYPE           }]    ]
[      [[ DESCENDING] {DATA-BASE-KEY         }]    ] . . .
```

In the DDL for multi-member-type sets, the following two elments consititute a difference in syntax from the DDL for mono-member-type sets:

1) the optional introduction of the member record-type as a sort-key item (keyword "RECORD-TYPE")

2) the definition of the sequence of reference of these record-types (RECORD-TYPE SEQUENCE Phrase).

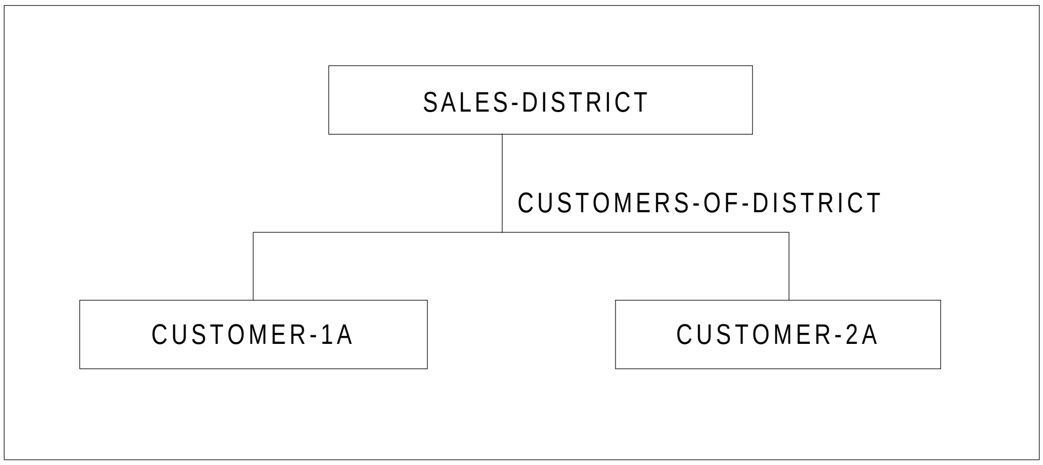
   - If RECORD-TYPE is not specified as a key item, the RECORD-TYPE SEQUENCE phrase must not be specified either. The same number of key items must be stated for each member record-type (one-to-one correspondence rule). Furthermore, each key item must be preceded by the same ASCENDING or DESCENDING keyword and have identical TYPE characteristics in all the member record-types; however its position within the record may vary from one record-type to another. (See Example 1, below.)
   - If RECORD-TYPE is specified as a key item, it must appear only once and the RECORD-TYPE SEQUENCE phrase must be present. See Examples 2 and 3, below.)

When key items are specified to the **left** of RECORD-TYPE, they must conform to the rules of one-to-one correspondence described directly above.

When key items are specified to the **right** of RECORD-TYPE, they may differ in number, TYPE characteristics, and in ASCENDING or DESCENDING keywords from one record-type to another. The general ordering of the key items, using fields common to all member-types, can then be followed by an ordering of the records of a given type, using fields specific to this record-type.

3.10.8.1 Example 1: RECORD-TYPE is not specified as a Key Item.

Suppose that two record types have been declared to represent a customer. The first record-type (CUSTOMER-1A) corresponds to a customer having 1 address, the second record-type (CUSTOMER-2A) corresponds to a customer having 2 addresses. These record-types are members of the same set (CUSTOMERS-OF-DISTRICT) and are sorted on the "customer-number", independently of their record-type.



The related DDL clause could be written as follows:

```
RECORD NAME IS CUSTOMER-1A
       ...
02     C-1A-NUMBER TYPE IS SIGNED PACKED DECIMAL 5.
02     C-1A-ADDRESS TYPE IS CHARACTER 35.
       ...
RECORD NAME IS CUSTOMER-2A
       ...
02     C-2A-NUMBER TYPE IS SIGNED PACKED DECIMAL 5.
02     C-2A-ADDRESS1 TYPE IS CHARACTER 35.
02     C-2A-ADDRESS2 TYPE IS CHARACTER 35.
       ...
SET NAME IS CUSTOMERS-OF-DISTRICT
    OWNER IS SALES-DISTRICTS
    ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS
                       DUPLICATES ARE NOT ALLOWED.

  MEMBER IS CUSTOMER-1A
       ...
    KEY IS ASCENDING C-1A-NUMBER.

  MEMBER IS CUSTOMER-2A
       ...
    KEY IS ASCENDING C-2A-NUMBER.
```

NOTE: The common sort-key item has the same position within each record-type, but this condition is not required.

After several insertions, an occurrence of the set CUSTOMERS-OF-DISTRICT would look like that illustrated by Figure 3-32.

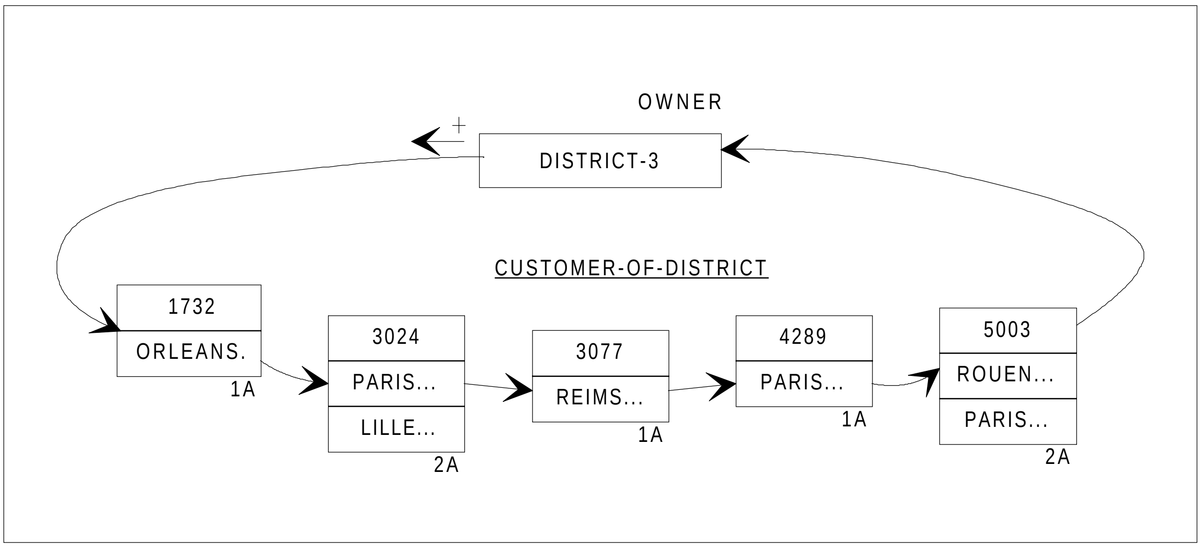
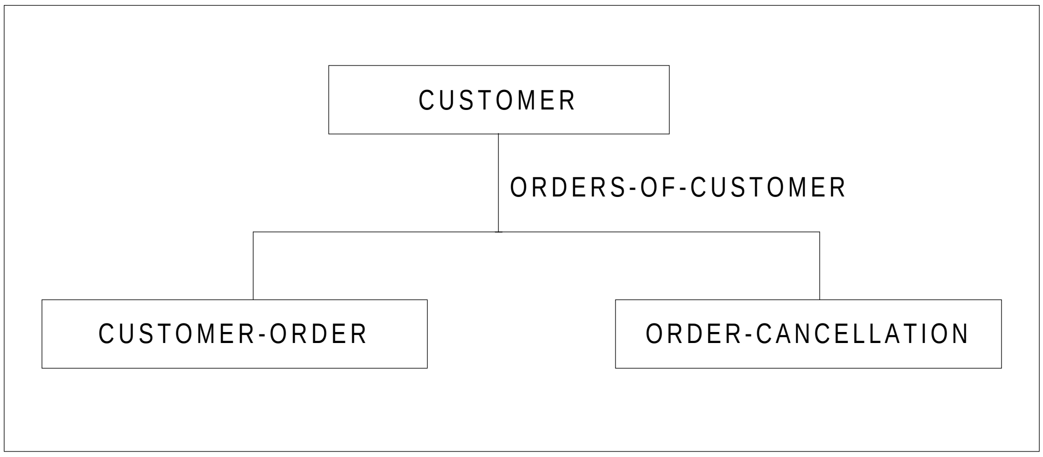


Figure 3-32. Order SORTED BY DEFINED KEYS (Multi-member-type Set); RECORD-TYPE is Not a Key Item

3.10.8.2 Example 2: RECORD-TYPE is the Last Key Item.

Suppose that the "ORDERS-OF-CUSTOMER" set contains the "CUSTOMER-ORDER" records and the "ORDER-CANCELLATION" records.



The "order-number" is assumed to appear in both record-types. If the set is to be sorted on the "order-number" and if an ORDER-CANCELLATION record must directly follow the corresponding CUSTOMER-ORDER record, the Schema DDL may be defined as follows:

```
RECORD NAME IS CUSTOMER-ORDER
       ...
02     ORDER-NUMBER TYPE IS SIGNED UNPACKED DECIMAL 7.
02     QUANTITY TYPE IS SIGNED BINARY 15.
02     ORDER-DATA TYPE IS CHRACTER 5.
       ...
RECORD NAME IS ORDER-CANCELLATION
       ...
02     CANCELLATION-DATE TYPE IS CHARACTER 5.
02     C-ORDER-NUMBER TYPE IS SIGNED UNPACKED DECIMAL 7.
       ...
SET NAME IS ORDERS-OF-CUSTOMER
    OWNER IS CUSTOMER
    ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS
             RECORD TYPE SEQUENCE IS CUSTOMER-ORDER
                                     ORDER-CANCELLATION
                    DUPLICATES ARE NOT ALLOWED.

  MEMBER IS CUSTOMER-ORDER
       ...
    KEY IS ASCENDING ORDER-NUMBER RECORD-TYPE.

  MEMBER IS ORDER-CANCELLATION
       ...
    KEY IS ASCENDING C-ORDER-NUMBER RECORD-TYPE.
```

They record-type is used to insert an ORDER-CANCELLATION record just after the CUSTOMER-ORDER record which contains the same "order-number".
After several insertions, a typical occurrence of set ORDERS-OF-CUSTOMER would look like that illustrated by Figure 3-33.

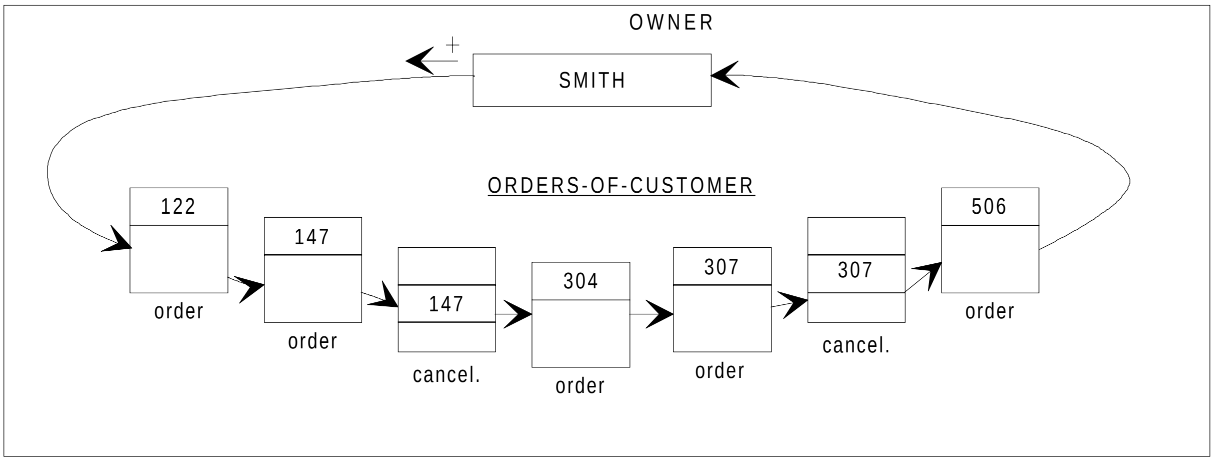
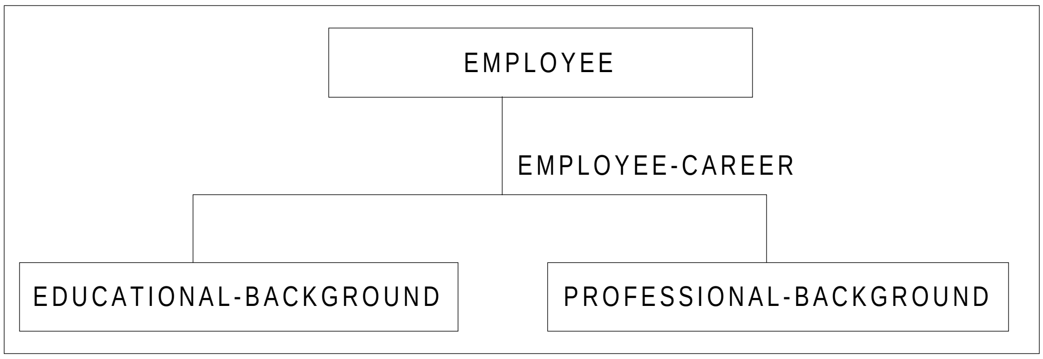


Figure 3-33. Order SORTED BY DEFINED KEYS (Multi-member-type set); RECORD-TYPE is the Last Key Item

3.10.8.3 Example 3: RECORD-TYPE is the First Key Item.

Suppose that the "EMPLOYEE-CAREER" set contains "EDUCATIONAL-BACKGROUND" records and "PROFESSIONAL-BACKGROUND" records.



If these three conditions must be met:

- the EDUCATIONAL-BACKGROUND records must precede all the PROFESSIONAL-BACKGROUND records,
- the EDUCATIONAL-BACKGROUND records must be sorted on the diploma-year
- and the PROFESSIONAL-BACKGROUND records must be sorted on the job-assignment-year and day.

The Schema DDL may be defined as follows:

```
RECORD NAME IS EDUCATIONAL-BACKGROUND
       ...
02     DIPLOMA-NAME TYPE IS CHARACTER 20.
02     DIPLOMA-YEAR TYPE IS CHARACTER 4.
02     UNIVERSITY-NAME TYPE IS CHARACTER 15.
       ...
RECORD NAME IS PROFESSIONAL-BACKGROUND
       ...
02     JOB-ASSIGNMENT-YEAR TYPE IS CHARACTER 4.
02     JOB-ASSIGNMENT-DAY TYPE IS CHARACTER 3.
02     JOB-ID TYPE IS CHARACTER 15.
02     COMPANY-NAME TYPE IS CHARACTER 25.
       ...
SET NAME IS EMPLOYEE-CAREER
    OWNER IS EMPLOYEE
    ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS
          RECORD TYPE SEQUENCE IS EDUCATIONAL-BACKGROUND
                                  PROFESSIONAL-BACKGROUND
                    DUPLICATES ARE LAST.

  MEMBER IS EDUCATIONAL-BACKGROUND
       ...
    KEY IS ASCENDING RECORD-TYPE DIPLOMA-YEAR

  MEMBER IS PROFESSIONAL-BACKGROUND
       ...
    KEY IS ASCENDING RECORD-TYPE JOB-ASSIGNMENT-YEAR
                                 JOB-ASSIGNMENT-DAY.
```

After several insertions, a typical occurrence of set EMPLOYEE-CAREER would look like that illustrated by Figure 3-34.

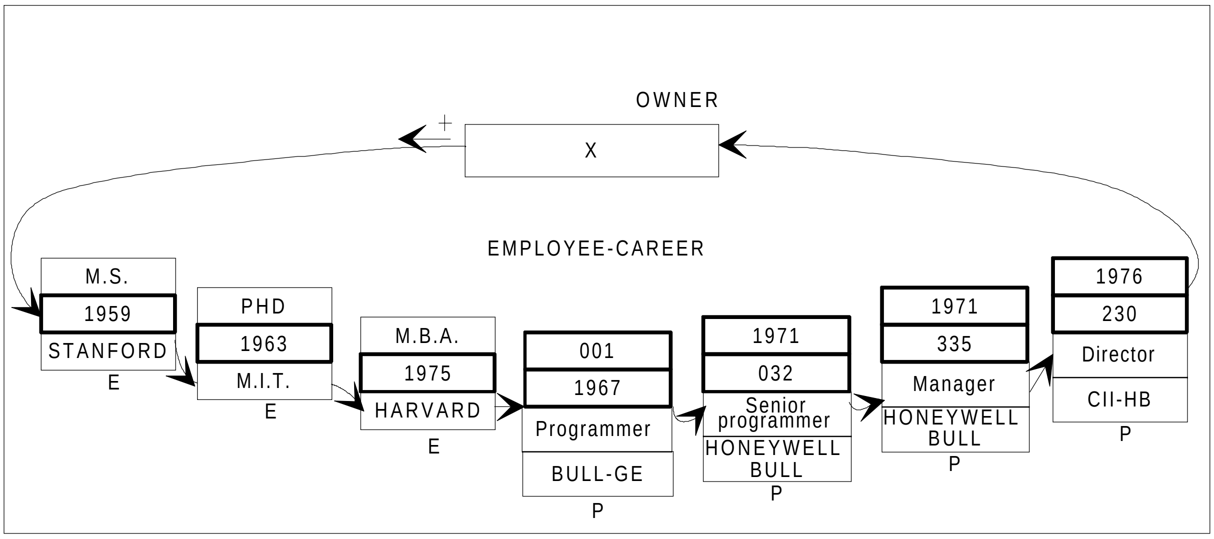


Figure 3-34. Order SORTED BY DEFINED KEYS (Multi-member-type Set); RECORD-TYPE Is the First Key Item

3.10.9 Order SORTED WITHIN RECORD-TYPE

The DDL syntax for a record to be inserted sorted within record-type is given below:

```
SET NAME IS . . .

    . . .

    ORDER IS PERMANENT INSERTION IS SORTED WITHIN RECORD-TYPE

(and for each member)

MEMBER IS . . .

[        {ASCENDING } {data-base-identifier-2}       ]
[ KEY IS {           } {RECORD-TYPE           }       ]
[        {DESCENDING} {DATA-BASE-KEY         }       ]
[                                                    ]
[      [[ ASCENDING ] {data-base-identifier-3}]      ]
[      [[           ] {RECORD-TYPE           }]  ..  ]
[      [[ DESCENDING] {DATA-BASE-KEY         }]      ]
[                                                    ]
        [                {FIRST              }]      ]
        [ DUPLICATES ARE {LAST               }]      ]
        [                {NOT ALLOWED        }]      ]
        [                {SYSTEM-DEFAULT     }]      ]
```

The insertion point of a record is determined only with respect to the other records of its type already in the set occurrence. Records of a different record-type are ignored.

The differences in DDL syntax from a record insertion SORTED BY DEFINED KEYS are listed below. With a record insertion SORTED WITHIN RECORD-TYPE:

- The key items are completely independent from one member-type to another. The key item RECORD-TYPE is not allowed.
- The DUPLICATES phrase is specified in the KEY Clause instead of the ORDER Clause and may differ from one KEY Clause to another.
- The KEY clause is optional, which means that records of one member-type may be sorted where as records of a different member-type are inserted in an undefined order. Nevertheless, the KEY Clause must appear in at least one Member Subentry to justify the SORTED phrase in the SET Entry.

This ordering method is used in certain cases where record-types, with no relationship to each other, are not declared as members of separate mono-record-type sets but are grouped in the same multi-record-type set, as shown on Figure 3-35.

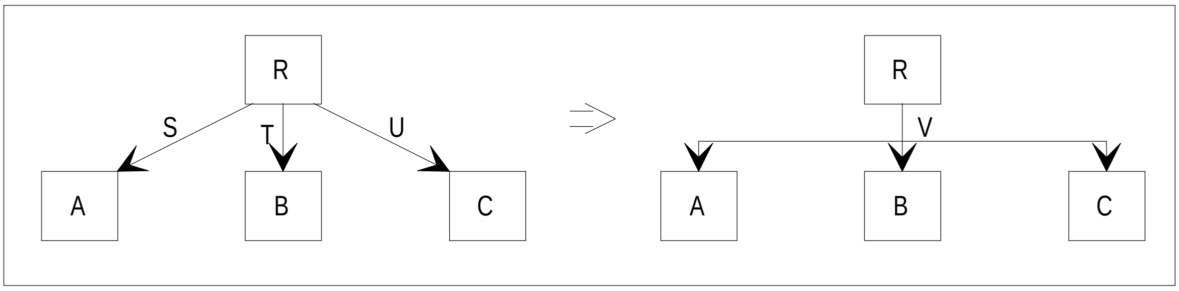


Figure 3-35. Several Sets Versus Multi-member-type-set

If the R occurrences have few and sometimes no members of type A, B, or C, then by grouping these members into one set saves the NEXT and PRIOR pointers in record R for two sets out of the three.

3.10.9.1 Example

Figure 3-36, further on in this subsection, illustrates the SORTED WITHIN RECORD-TYPE method. The related DDL clauses are:

```
 SET NAME IS V
     OWNER IS R
     ORDER IS PERMANENT INSERTION IS SORTED WITHIN
                                     RECORD-TYPE.

   MEMBER IS A
        ...
     KEY IS DESCENDING A-DATE DUPLICATES ARE FIRST
        ...
   MEMBER IS B
        ...
     KEY IS ASCENDING B-NAME DUPLICATES ARE LAST
        ...
   MEMBER IS C
        ...
```

The relative position of records of different types can be clarified by explaining how the DBCS proceeds for its comparisons. The search is performed in the forward direction:

- 1st insertion: There are no other records in the set at this point.
- 2nd insertion: the DBCS skips A7 to find a possible B record. Since there is none, B8 is inserted between A7 and R3.
- 3rd insertion: the DBCS compares 1974 with 1976 of A7. As it is less than 1976, the DBCS goes on, skipping B8 and inserts A3 between B8 and R3.
- 4th insertion: the DBCS skips A7, finds B8 whose key "JOHN" is higher than "JANE". It inserts B5 just before B8.
- 5th insertion: record C has no KEY Clause in its MEMBER Subentry. The DBCS inserts it according to an ORDER FIRST (between R3 and A7).
- 6th insertion: the DBCS skips C6, goes beyond A7 since 1974 is less than 1976, skips B5 and B8, and then finds A3. Since the KEY Clause of record A indicates that DUPLICATES ARE FIRST, the DBCS inserts A6 just before A3.
- 7th insertion: the DBCS skips C6 and A7, goes beyond B5 since its key "JANE" is lower than "JOHN", and finds B8. Since the KEY Clause of record B indicates that DUPLICATES ARE LAST, the DBCS goes beyond B8, skips A6 and A3, and inserts B7 between A3 and R3.

The main purpose of this detailed explanation is is to demonstrate that the SORTED WITHIN RECORD-TYPE order does not result in the records being grouped by type along the set occurrence.

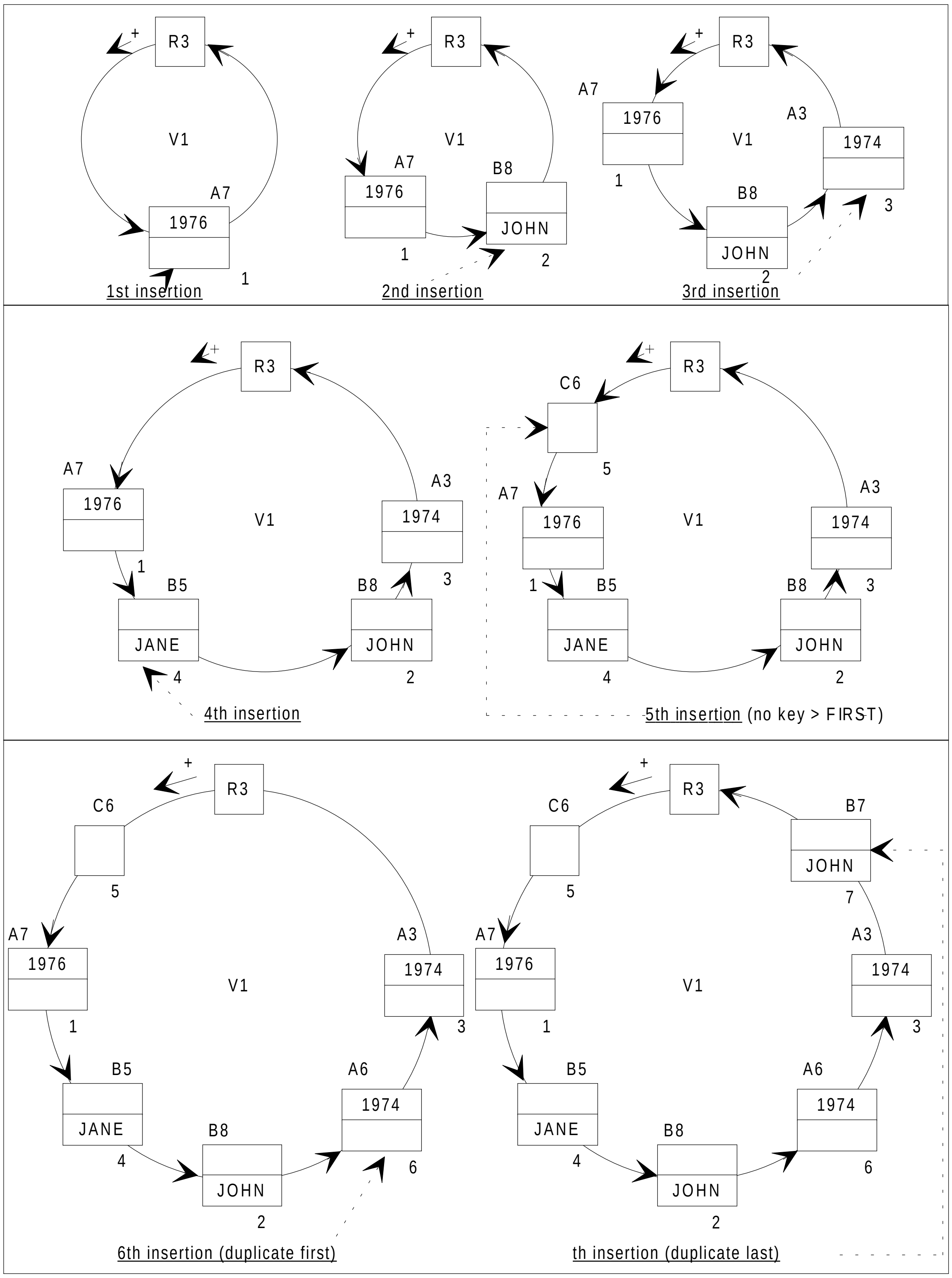


Figure 3-36. Order SORTED WITHIN RECORD-TYPE

3.10.10 Performance Aspects in terms of the Sorted Order Chosen

The connection of a record to a sorted set requires a search of the set occurrence in the forward direction up to the insertion point. The starting point of this search depends on the set selection criteria and influences the performance of the insertion process. The difference between the final SORTED order and the chronological order of insertion also affects performance.

3.10.10.1 Optimization for a Set Selection Path with Only one Level Identified by APPLICATION

In this case, the DBCS takes into account the **current-of-set**:

- If the current-of-set is the owner, the starting point is the owner.
- If the current-of-set is a member of any type for a set SORTED BY DEFINED KEYS or of the same type for a set SORTED WITHIN RECORD-TYPE, the key of the new member is compared with the key of the current-of-set as follows:
  - If the new member is to be inserted after the current-of-set, the starting-point is the current-of-set.
  - If the new member is to be inserted before the current-of-set, the starting-point is the owner.
- If the current-of-set is of a different type for a set SORTED WITHIN RECORD-TYPE, the starting-point is the owner.

For this type of set selection path, the insertion process is optimized when members are **chronologically** inserted, in one of the two ways mentioned below:

- in the reverse order of the SORTED order (insertion FIRST just between the owner and the current-of-set)
- or in the same order as the SORTED order (insertion LAST just between the current-of-set and the owner)

3.10.10.2 Optimization for Another Type of Set Selection Path

The starting point of the search is always the owner.

The insertion process is optimized when members are **chronologically** inserted in the **reverse order** of the SORTED order (insertion FIRST). By contrast, a chronological order of insertion equal to the SORTED order results in the worst possible performance (insertion LAST after a search of the whole occurrence).

3.10.11 Absence of Optimization

When no optimization is possible in the program logic, the SORTED order should be restricted to sets whose occurrences are short and clustered.

3.11 "NO DUPLICATES" CONTROL

3.11.1 Purpose

The DBCS may be requested to check that some records do not contain the same values in certain fields.

The **scope** of the "NO DUPLICATES" control and the **point in time** when this check is performed depend on the function of the elementary field being checked: 1) CALC-key within an area, 2) sort-key or 3) other fields within a set occurrence. These cases are explained in the subsections directly below.

3.11.2 CALC-key within an Area

The "NO DUPLICATES" control is requested in the LOCATION Clause of the RECORD Entry, as shown below.

```
LOCATION MODE IS CALC USING {identifier}...
         DUPLICATES ARE NOT ALLOWED
```

When storing a record of this type into an area or when modifying the CALC-key of this record, the DBCS checks that the value of the CALC-key is different from the value iof the CALC-key of records of **this type** already present in **this area**.

3.11.3 Sort-key within a Set Occurrence

In this case, the "NO DUPLICATES" control is requested in the ORDER Clause of the SET Entry if the set is SORTED BY DEFINED KEYS. This Clause is shown below.

```
(sets)
       ORDER IS PERMANENT
       INSERTION IS SORTED BY DEFINED KEYS
       ...
       DUPLICATES ARE NOT ALLOWED

(members)
       KEY IS ...
```

If the set is SORTED WHITHIN RECORD-TYPE, the "NO DUPLICATES" control is requested in the KEY Clause of the MEMBER Subentry. This Clause is shown below.

```
(sets)

       ORDER IS PERMANENT

       INSERTION IS SORTED WITHIN RECORD-TYPE

       ...


(members)

       KEY IS ...DUPLICATES ARE NOT ALLOWED
```

When a member record of this type is inserted into an occurrence of the corresponding set-type (or reinserted in certain cases of MODIFY), the DBCS checks that the value of its sort-key (composed of all the key items) is different from that of the members already present in **this set occurrence**.

If the set has one member-type or if it is SORTED WITHIN RECORD-TYPE, the "NO DUPLICATES" control applies to **the member-type considered**.

If the set has several member-types and is SORTED BY DEFINED KEYS, the "NO DUPLICATES" control applies to the corresponding sort-keys of **all the member-types**. Note that if RECORD-TYPE is a key item, the control is in fact applied to each record-type separately.

3.11.4 Other Fields within a Set Occurrence

The "NO DUPLICATES" control may be requested for any group of elementary fields with the DUPLICATES Clause of the MEMBER Subentry, which is shown below.

```
MEMBER IS ...
       {DUPLICATES ARE NOT ALLOWED FOR {identifier}...}...
```

Each clause applies to a field or to a group of fields of the record. Several clauses may be written if several groups are to be checked.

Note that the following series of clauses

```
DUPLICATES ARE NOT ALLOWED FOR A
DUPLICATES ARE NOT ALLOWED FOR B
DUPLICATES ARE NOT ALLOWED FOR C
```

is more restrictive than the clause

```
DUPLICATES ARE NOT ALLOWED FOR A B C
```

In terms of the clauses above, assuming that A, B and C are numeric, the record with key values **A=5**, **B=2**, and **C=3** and the record with the key values **A=4**, **B=2**, and **C=6** are considered as duplicates in the first case defined by the first series (because of B=2) and non-duplicates in the second series.

When inserting a record of this type into an occurrence of the corresponding set or when modifying the values of some of the fields under control, the DBCS checks that the values of the groups of fields are different from those of **the members of this type** already present in **this set occurrence**.

Figure 3-37, on the following page, illustrates the scope and time of execution of the "NO DUPLICATES" control for record data items.

RECORD DATA ITEMS UNDER CONTROL	**SCOPE OF "NO DUPLICATES" CONTROL**		**TIME WHEN CONTROL IS EXECUTED**
CALC-KEY	record-type	area	storage modification
SORT-KEY (DEFINED KEYS)	all member-types	Set Occurrence	insertion reinsertion (after modification)
SORT-KEY (WITHIN RECORD-TYPE)	member-type	Set Occurrence	insertion reinsertion (after modification)
groups of other fields	member-type	Set Occurrence	insertion modification

Figure 3-37. "NO DUPLICATES" Control

3.11.5 Performance Aspects

The database administrator must be aware that the "NO DUPLICATES" control is more or less time consuming depending on the length of the search.

- For CALC-keys, the DBCS must search the "CALC-chain" determined by the randomization.
- For sort-keys, the "NO DUPLICATES" control is included in the processing which determines the insertion (or reinsertion) point in the set occurrence. Therefore it does not require much time.
- For groups of other fields, the control implies a search of the entire set occurrence. This is the case when the record is inserted into a set occurrence or when a MODIFY statement alters some of these fields.

3.12 VALIDITY CHECKS ON DATA ITEMS

3.12.1 Purpose

For a given record-type, the DBCS may be requested to run validity checks to verify:

- that the contents of a data item belong to permitted ranges of values
- that conditions involving several data items of the record are satisfied.

These checks are performed when the record is stored into the database (STORE function) or when some of its fields are modified (MODIFY function).

3.12.2 Check Involving one Data Item (Data Level)

This check is defined in the Data Subentry of the Schema DDL:

```
 CHECK IS VALUE [NOT] literal-1 [THRU literal-2]  ...
```

If the THRU phrase is omitted, literal-1 defines a discrete value. If the THRU phrase is specified, literal-1 and literal-2 define a range of values including the **boundaries** (see **NOTE** below).

Literals must be specified in ascending order, according to the sequence of reference of their type.

If NOT is omitted, the value of the data item is valid if it is equal to one of the discrete values or belongs to one of the ranges of values.

If NOT is specified, the value of the data item is valid if it is neither equal to one of the discrete values nor belongs to one of the ranges of values.

NOTE: The TYPE Clause implicity defines boundaries for the values of a data item since it specifies a maximum number of characters or digits. The CHECK Clause is used to further restrict the values permitted within these implicit boundaries.

Examples:

```
04 PRODUCT-CODE   TYPE CHAR 1           CHECK VALUE "A" "B" "C" .
02 PERSON-AGE     TYPE UNPACKED DEC 2    CHECK VALUE 18 THRU 65.
```

3.12.3 Check Involving several Data Items (Record Level)

This check is defined in the Record Subentry of the Schema DDL:

```
CHECK  IS  condition
```

The condition is expressed as in COBOL by a combination of relation conditions, logical relators (AND, OR, NOT) and parentheses. Unlike COBOL, however, arithmetic expressions and abbreviated combined relation conditions are not allowed.

There may be several CHECK Clauses in the Record Subentry. The same data item can be the subject of a CHECK Clause in the Data Subentry and of one or several CHECK Clauses in the Record Subentry.

Example:

```
RECORD  PERSON
    LOCATION ...
    WITHIN ...
    CHECK DEATH-DATE >= BIRTH-DATE
    CHECK   SEX = "M" AND MAIDEN-NAME = " "
         OR SEX = "F" AND MARRIED = "N" AND MAIDEN-NAME = " "
         OR SEX = "F" AND MARRIED = "Y" AND NOT (MAIDEN-NAME = " ").
02 BIRTH-DATE TYPE UNPACKED DEC 4.
02 DEATH-DATE TYPE UNPACKED DEC 4.
02 SEX  TYPE CHAR 1.
02 MARRIED TYPE CHAR 1    CHECK VALUE "Y" "N".
02 MAIDEN-NAME TYPE CHAR 30.
```

3.13 GENERAL FORMAT OF THE SCHEMA DDL

The general format of the Schema DDL is given below:

```
                SCHEMA DDL

SCHEMA entry.


AREA                 Entry ...


RECORD               Entry ...
  LOCATION           Clause
  [CHECK             Clause]
  [KEY               Clause] ...
  [DATA              Subentry] ...
     TYPE            Subentry
     [CHECK          Clause]
     [DEFAULT        Clause]


[SET                 Entry  ] ...
  OWNER              Clause
  ORDER              Clause

  MEMBER             Subentry ...
     INSERTION       Clause
     RETENTION       Clause
     [DUPLICATES     Clause]
     [KEY            Clause]
     SET SELECTION   Clause



[KEY                 Entry  ] ...
     [USING          Clause]
     [WITHIN         Clause]
     [DUPLICATES     Clause]
```

Figure 3-38. General Format of the Schema DDL

4. Subschema Model

4.1 THE CONCEPT OF THE SUBSCHEMA

A **schema** usually has more data types in its definition than any application program would need to use. If a schema undergoes a change, all application programs which use this schema must be recompiled, even if the programs not logically impacted.

Certain data types are not visible to some application programs, or else they can be seen but not modified for security or confidentiality reasons. Other data types which are useful for one application program are not adapted for another one. Other problems can arise from the order in which fields are declared in a record type.

The naming of schema objects can also be specific to an application for methodological reasons.

These are all problems of scope and definition which have been solved by the creation of the concept of the **subschema**.

4.2 THE CONSTITUTION OF A SUBSCHEMA

A subschema describes a **partial view** of the database. This partial view of the database is made available, without any effect on the rest of the database, to a set of application programs.

An application program can use up to **thirty-two** subschemas. A subschema can be used by an arbitrary number of application programs. However, an application program cannot use two subschemas based on the same schema.

The general format of the Subschema Data Description Language (SDDL) is given below:

```
                    SUBSCHEMA DDL

      SUBSCHEMA entry.


      [ALIAS                 Entry  ]  ...


      AREA                   Entry  ...


      RECORD                 Entry  ...
        [ACCESS RESTRICTION Clause]
        [WITHIN AREA         Clause]
        [DATA                Subentry]  ...
           TYPE              Subentry   ...
           [ACCESS RESTRICTION Clause]

      [SET                   Entry  ]  ...
        [ACCESS RESTRICTION Clause]
        MEMBER               Subentry  ...
           [ACCESS RESTRICTION Clause]
           [SET SELECTION   BY APPLICATION]
```

There are six different mechanisms for defining a partial view of the database based on the schema. These are mechanisms which are used for writing a subschema and are listed below; they are further described in the subsections which follow.

- INCLUSION (and EXCLUSION) of schema objects from the subschema
- RENAMING of schema objects
- DATA TRANSFORMATIONS of schema field types
- ACCESS RESTRICTION on the data manipulation verbs of objects schemas
- REORDERING of fields
- the redefining of SET SELECTION

The sub-schema cannot be used to create a new object. It is always used to create a partial view of the database based on the schema.

4.2.1 The Inclusion Mechanism

A schema object becomes part of the subschema by including the **Schema DDL name of that object** in the corresponding SDDL entry.

Example:

```
                   SUBSCHEMA DDL
AREA   NAME   IS A01.
AREA   NAME   IS A02.

RECORD NAME   IS R01.    ( R01 is included with
                           all its fields     )
RECORD NAME   IS R02.
 01    R02F05.           ( R02 is included
                           with this field only )

SET    NAME   IS S01.    ( S01 is included with
                           all its members    )
SET    NAME   IS S03.
       MEMBER IS R02.    ( S03 is included
                           with this member only )

KEY    NAME   IS K02.
```

Schema objects can, by the same token, be **excluded** from the subschema.

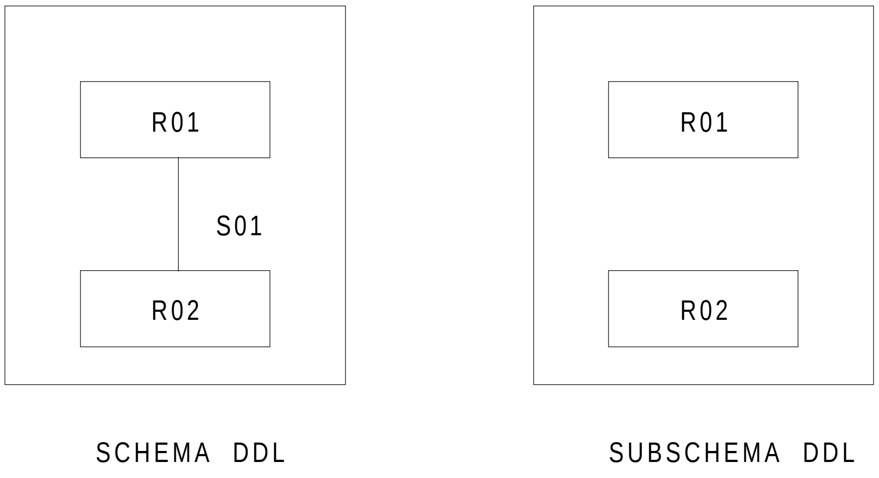
Exclusion of an object has effects on subschema objects, which are explained in the subsection below.

4.2.1.1 Set Exclusion

Any set named in the Schema DDL can be excluded from the Subschema DDL.

Example:

A graphic representation of the exclusion of set S01 is given below:



The corresponding Schema DDL source is shown below:

```
                   SCHEMA DDL
RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
01  R01F01  TYPE CHAR 3.
01  R01F02  TYPE SIGNED BINARY 15.

RECORD NAME IS R02
    LOCATION MODE IS DIRECT DB-PARAM2
    WITHIN ANY AREA.
01  R02F01  TYPE CHAR 8.

SET NAME IS S01
    OWNER IS R01
    ORDER IS PERMANENT INSERTION IS LAST.
MEMBER IS R02
    INSERTION IS AUTOMATIC RETENTION IS MANDATORY
    SET SELECTION FOR S01 IS
     THRU S01 OWNER IDENTIFIED BY CALC-KEY.
```

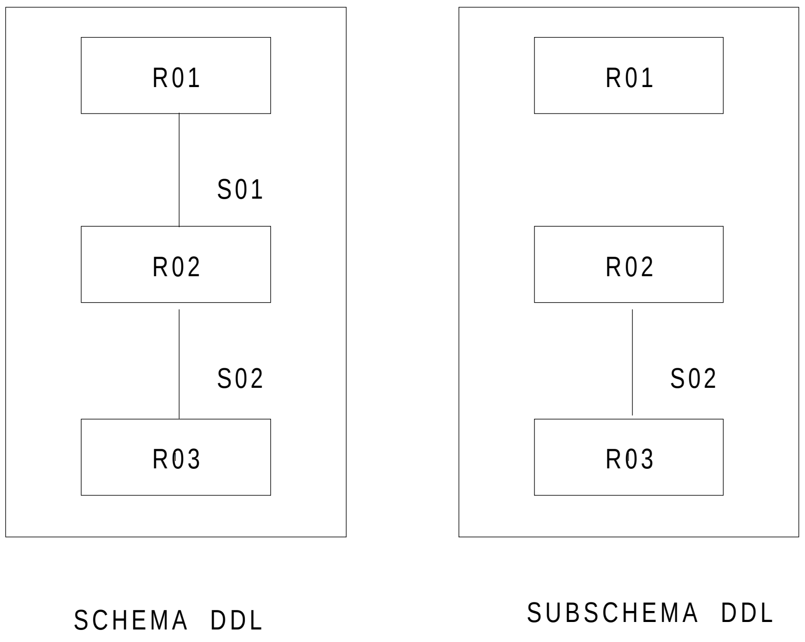
The corresponding Subschema DDL source is:

```
                 SUBSCHEMA DDL
RECORD NAME IS R01.

RECORD NAME IS R02.
```

All the SET SELECTION Clauses which reference an excluded set must be redefined BY APPLICATION.

Example:

Exclusion of set S01 affects the set selection for set S02:



The corresponding Schema DDL source is:

```
                    SCHEMA DDL
  SET NAME IS S02
      OWNER IS R02
      ORDER IS PERMANENT INSERTION IS LAST.
  MEMBER IS R03
      INSERTION IS AUTOMATIC RETENTION IS MANDATORY
      SET SELECTION FOR S02 IS
       THRU S01 OWNER IDENTIFIED BY CALC-KEY
       THEN THRU S02 WHERE OWNER IDENTIFIED BY R02F01
```

The corresponding Subschema DDL source is:

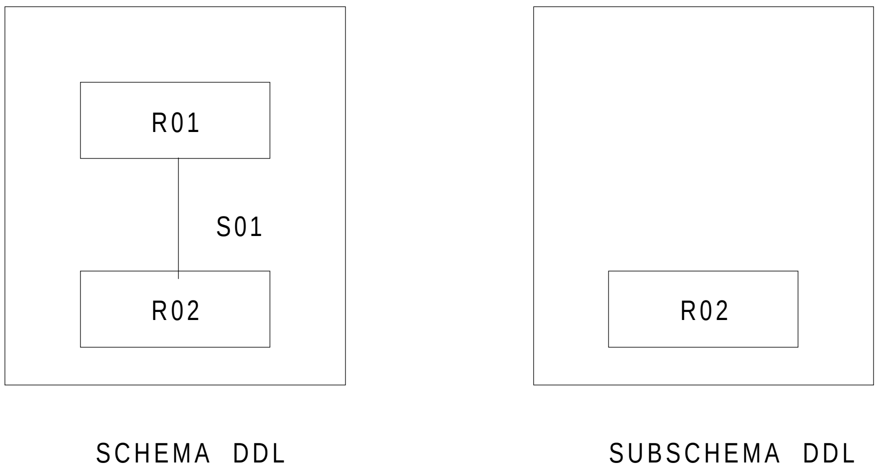
```
                      SUBSCHEMA DDL

  SET NAME IS S02

  MEMBER IS R03

      SET SELECTION FOR S02 IS

       THRU S02 OWNER IDENTIFIED BY APPLICATION.
```

4.2.1.2 Record Exclusion

If the excluded record is defined as the owner of a set in the Schema DDL, the set must be excluded from the subschema. The same occurs if the excluded record is the only member of a set.

Example:

Exclusion of record R01:



The corresponding Subschema DDL source is:

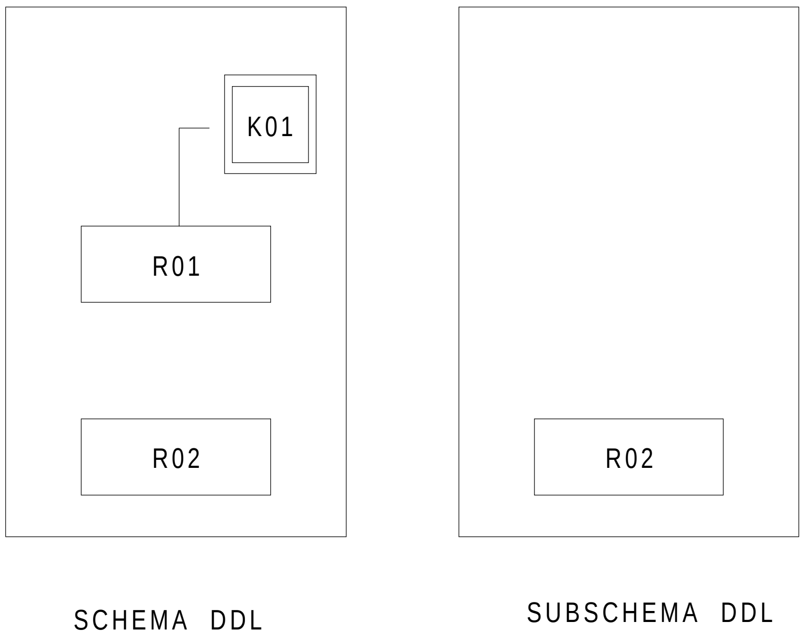
```
                  SUBSCHEMA DDL

  RECORD NAME IS R02.
```

Set S01, which no longer has an owner (its owner was R01), must be excluded from the Subschema

If a mono-record-type key was defined on a record in the Schema DDL, that key must also be excluded from the subschema.

Example:

Exclusion of record R01:



The corresponding Schema DDL source is:

```
                      SCHEMA DDL
RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
KEY NAME IS K01
    USING ASCENDING R01F02.

01  R01F01  TYPE CHAR 3.
01  R01F02  TYPE SIGNED BINARY 15.

RECORD NAME IS R02
    LOCATION MODE IS DIRECT DB-PARAM2
    WITHIN ANY AREA.
01  R02F01  TYPE CHAR 8.

KEY NAME IS K01
    USING DB-PARAM-1
    WITHIN ANY AREA
DUPLICATES ARE NOT ALLOWED.
```

The corresponding Subschema DDL source is:

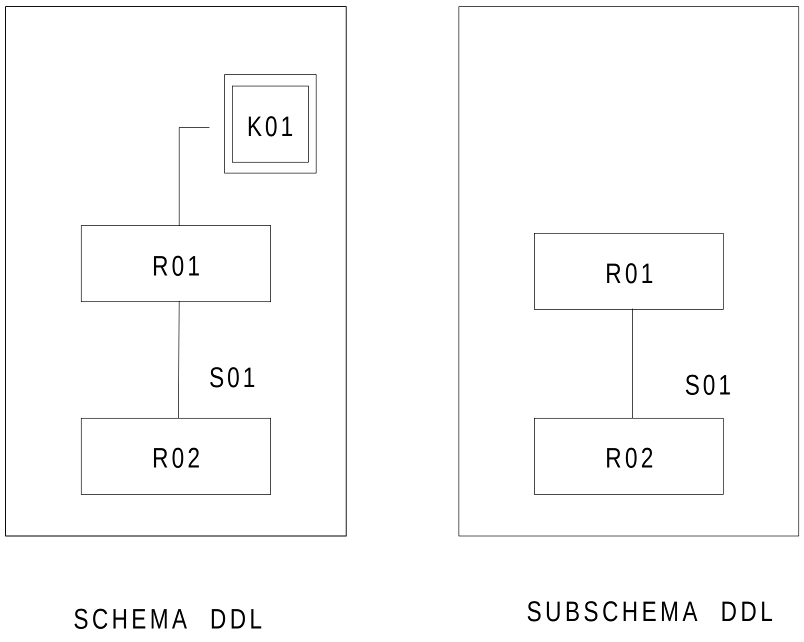
```
                    SUBSCHEMA DDL

RECORD NAME IS R02.
```

4.2.1.3 Key Exclusion

A key can be excluded from the Subschema DDL, but if a SET SELECTION Clause uses this key in the Schema DDL, the set- selection must be redefined BY APPLICATION.

Example:

Exclusion of key K01:



The corresponding Schema DDL source is:

```
                    SCHEMA DDL

RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
KEY NAME IS K01
    USING ASCENDING R01F02.

01  R01F01  TYPE CHAR 3.
01  R01F02  TYPE SIGNED BINARY 15.

RECORD NAME IS R02
    LOCATION MODE IS DIRECT DB-PARAM2
    WITHIN ANY AREA.
01  R02F01  TYPE CHAR 8.

KEY NAME IS K01
    USING DB-PARAM-1
    WITHIN ANY AREA
DUPLICATES ARE NOT ALLOWED.
SET NAME IS S01
    OWNER IS R01
    ORDER IS PERMANENT INSERTION IS LAST.
MEMBER IS R02
    INSERTION IS AUTOMATIC RETENTION IS MANDATORY
    SET SELECTION FOR S01 IS
     THRU S01 OWNER IDENTIFIED BY KEY K01.
```

The corresponding Subschema DDL source is:

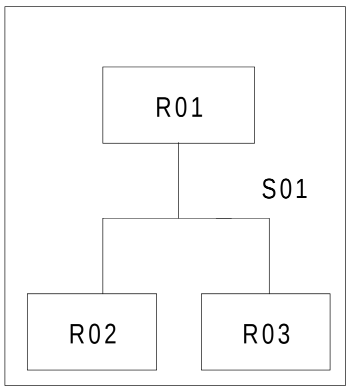
```
                      SUBSCHEMA DDL

RECORD NAME IS R01.
RECORD NAME IS R02.
SET NAME IS S01
    SET SELECTION FOR S01 IS
     THRU S01 OWNER IDENTIFIED BY APPLICATION.
```

4.2.1.4 Member Exclusion

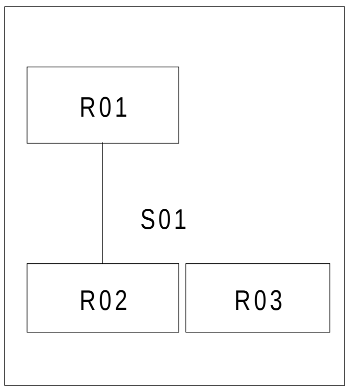
A member of a set can be excluded from the subschema. If the member to be excluded is the only member of the set, that set must also be excluded from the subschema.

Example:

Exclusion of member R03 (but not of record R03)



SCHEMA DDL



SUBSCHEMA DDL

The corresponding Schema DDL source is:

```
                    SCHEMA DDL

RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.

01  R01F01  TYPE CHAR 3.
01  R01F02  TYPE SIGNED BINARY 15.

RECORD NAME IS R02
    LOCATION MODE IS DIRECT DB-PARAM2
    WITHIN ANY AREA.
01  R02F01  TYPE CHAR 8.

RECORD NAME IS R03
    LOCATION MODE IS CALC USING R03F01
    WITHIN ANY AREA
DUPLICATES ARE  ALLOWED.

SET NAME IS S01
    OWNER IS R01
    ORDER IS PERMANENT INSERTION IS LAST.
MEMBER IS R02
    INSERTION IS AUTOMATIC RETENTION IS MANDATORY
    SET SELECTION FOR S01 IS
     THRU S01 OWNER IDENTIFIED BY APPLICATION
MEMBER IS R03
    INSERTION IS AUTOMATIC RETENTION IS MANDATORY
    SET SELECTION FOR S01 IS
     THRU S01 OWNER IDENTIFIED BY APPLICATION.
```

The corresponding Subschema DDL source is:

```
                     SUBSCHEMA DDL



RECORD NAME IS R01.

RECORD NAME IS R02.

RECORD NAME IS R03.



SET NAME IS S01

    MEMBER IS R02.
```

4.2.1.5 Field Exclusion

As we have seen, a field can be excluded from a record if that record is defined in the subschema with a restricted list of fields.

Take the following schema description of record R01 as an example:

```
                SCHEMA DDL
RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
01  R01F01  TYPE CHAR 4.
01  R01F02  TYPE SIGNED BINARY 15.
01  RO1F03.
 02 R01F04  TYPE IS UNSIGNED UNPACKED DECIMAL 4,1.
 02 R01F05  TYPE IS SIGNED BINARY OCCURS 6.
01  R01F06  OCCURS 10 DEPENDING ON RO1F02.
 02 R01F07  TYPE CHAR 8.
 02 R01F08  TYPE UNSIGNED PACKED DEC 4,-1.
```

The following subschema description of record R01 is an example of group exclusion:

```
                 SUBSCHEMA DDL
RECORD NAME IS R01.
01  R01F01  .
01  R01F02  .

                   (exclusion of group R01F03)

01  R01F06  .
 02 R01F07  .
 02 R01F08  .
```

What is important to note is that, when a group is excluded from the subschema, all the fields belonging to that group must also be excluded. On the other hand, a field belonging to a group can be excluded even if that group is not excluded.

Example:

```
                   SUBSCHEMA DDL
RECORD NAME IS R01.
01  R01F01  .
01  R01F02  .

                   (exclusion of group R01F03)

01  R01F06  .

                   (exclusion of field R01F07
                    from the repeating group)

 02 R01F08  .
```

If a control field is excluded from the Schema DDL, control fields must also be excluded.

In this context, control field is a field whose name appears at least twice in the Schema DDL.

Examples of control fields are given below:

- a CALC field
- a field within a Check Clause
- a key field
- a field DUP NOT within a set .
- a field of a depending on Clause
- a field of a SET SELECTION Clause

To take an example, record R01, shown in the Schema DDL above, has 2 control fields:

- R01F01 (CALC field)
- R01F02 (DEPENDING ON field)

If R01F02 is excluded, the OCCURS DEPENDING ON group R01F06 must also be excluded. Nevertheless, the OCCURS DEPENDING ON group R01F06 can be excluded **without** the associated field R01F02 being excluded. This creates a case where R01 is a variable length record in the schema but a **fixed length record** in the subschema.

4.2.2 Renaming a Schema DDL Object

Any Schema DDL object can be renamed by using the ALIAS Clause, which is shown below:

```
  ALIAS OF OBJECT   DDL-OBJECT-NAME IS SDDL-OBJECT-NAME
```

Examples:

```
ALIAS OF AREA         ddl-area   IS    SDDL-area
ALIAS OF RECORD       ddl-record IS    SDDL-record
ALIAS OF FIELD        ddl-field  OF    ddl-record  IS   sddl-field
ALIAS OF SET          ddl-set    IS    sddl-set
ALIAS OF KEY          ddl-key    IS    sddl-key
ALIAS OF PARAMETER    ddl-param  IS    sddl-param
```

4.2.3 Data Transformations

It is possible to redefine the type of a data item in accordance with the conversion rules summarized in the diagram below. Examples are given directly following the diagram.

	SDDL type			
DDL type	DECIMAL m,p	BINARY 15	BINARY 31	CHAR n
DECIMAL m,p	YES (1) (2)	YES 1	YES (1)	
BINARY 15	YES (1) (2)	YES	YES (1)	
BINARY 31	YES (1) (2)	YES (1)	YES	
CHARACTER n				YES (1)

Figure 4-1. DDL and SDDL Data Type Conversions

Meaning of notes in the table:
(1) possible truncation and/or loss of precision
(2) possible loss of minus sign

Examples:

An example of a data transformation defined in the Schema DDL is given below:

```
RECORD NAME IS R ...
01      A   TYPE IS UNSIGNED UNPACKED DECIMAL 8,2.
01      B   TYPE IS SIGNED PACKED DECIMAL 4.
 02     C.
  03    D   TYPE CHAR 7.
  03    E   TYPE IS SIGNED BINARY 31.
```

And in the corresponding Subschema DDL:

```
RECORD NAME IS R ...
01      A   TYPE IS UNSIGNED UNPACKED DECIMAL 6,2.
                               (truncation)
01      B   TYPE IS UNSIGNED PACKED DECIMAL 4.
                               (loss of minus sign)
 02     C.
  03    D   TYPE IS SIGNED BINARY 15.
                               (conversion not allowed)
  03    E   TYPE IS SIGNED BINARY 15.
                               (loss of precision)
```

NOTE: The conversion of a data type is not allowed for a data item which is used as a control field in the Schema DDL.

4.2.4 Access Restriction

4.2.4.1 Restrictions on Areas

The usage mode of an area can be restricted by specifying the following Subschema DDL clause:

```
AREA NAME IS area-name-1

     [ { | RETRIEVAL |                    }        ]
     [ { | UPDATE    |        {  ALLOWED  } }        ]
     [ { | SHARED    |   IS  {            } }   . . . ] .
     [ { | MONITORED |        {NOT ALLOWED} }        ]
     [ { | EXCLUSIVE |                    }        ]
```

4.2.4.2 Restrictions on Records

The execution of a DDL manipulation verb on a record can be restricted by specifying the following Subschema DDL clause:

```
RECORD NAME IS record-name-1

       [ { | STORE  |                   }       ]
       [ { | GET    |     {  ALLOWED  } }       ]
       [ { | MODIFY |  IS {           } } . . . ]
       [ { | ERASE  |     {NOT ALLOWED} }       ]
       [ { | FIND   |                   }       ]

[ WITHIN AREA area-name-1 [,area-name-2] . . .  ] .
```

The WITHIN Clause specifies the restricted list of areas in which the record is visible via the subschema definition.

4.2.4.3 Restrictions on Fields

The verbs GET or MODIFY, when directed to a field, can also be restricted.

Examples:

In the Schema DDL:

```
  RECORD NAME IS R ...
   01     A   OCCURS 3.
    02    B   TYPE CHAR 2.
    02    C.
     03   D   TYPE CHAR 7.
     03   E   TYPE CHAR 30 OCCURS 2.
```

In the Subschema DDL:

```
RECORD NAME IS R ...
01       A.
  02     B   GET IS NOT ALLOWED.
  02     C.
    03   D   MODIFY IS NOT ALLOWED.
    03   E.
```

Restriction on records and fields must be coherent. Examples of DDL which are coherent (therefore authorized) and incoherent are given below.

Examples:

The following Subschema DDL is authorized:

```
RECORD NAME IS R ...   GET    IS NOT ALLOWED
                       MODIFY IS NOT ALLOWED.
01       A.
  02     B   GET IS ALLOWED.
  02     C.
    03   D   MODIFY IS ALLOWED.
    03   E.
```

The following Subschema DDL is forbidden:

```
RECORD NAME IS R ...   GET    IS ALLOWED
                       MODIFY IS ALLOWED.
01       A.
  02     B   GET IS NOT ALLOWED.          <===FORBIDDEN
  02     C.
    03   D   MODIFY IS NOT ALLOWED.       <===FORBIDDEN
    03   E.
```

4.2.4.4 Restrictions on Members

The DDL manipulation verbs FIND, CONNECT, DISCONNECT or MODIFY applied to a member of a set can also be restricted, by using the following statement:

```
MEMBER IS record-name-1

  [ { | FIND       |    {  ALLOWED  } }       ]
  [ { | CONNECT    |    {           } }       ]
  [ { | DISCONNECT | IS {           } } . . . ]
  [ { | MODIFY     |    {NOT ALLOWED} }       ]
```

4.2.5 Reordering of Fields

The relative order of the data description determines the relative order of data items in the UWA record. It is possible to modify the relative order of two fields in relationship to the relative order defined in the Schema DDL if these fields are members of the same groups in the Schema DDL.

Examples:

In the Schema DDL:

```
01       A   OCCURS 3.
  02     B   TYPE CHAR 2.
  02     C.
    03   D   TYPE CHAR 7.
    03   E   TYPE CHAR 30 OCCURS 2.
```

In the Subschema DDL, the following two examples of the reordering of fields is possible:

Example 1: *Transformation within group C*

```
01       A   OCCURS 3.
  02     B   TYPE CHAR 2.
  02     C.
    03   E   TYPE CHAR 30 OCCURS 2.
    03   D   TYPE CHAR 7.
```

Example 2*: Transformation within group C and between groups C and B (all within group A)*

```
01       A   OCCURS 3.
  02     C.
    03   E   TYPE CHAR 30 OCCURS 2.
    03   D   TYPE CHAR 7.
  02     B   TYPE CHAR 2.
```

However, the reordering of fields shown in the following example below is forbidden because a field cannot change its group membership. E belongs to group C and therefore cannot belong to group B.

```
01       A   OCCURS 3.
  02     B   TYPE CHAR 2.
    03   E   TYPE CHAR 30 OCCURS 2.      <===FORBIDDEN
  02     C.
    03   D   TYPE CHAR 7.
```

VARIABLE RECORD LENGTH CASE:A DEPENDING ON field must be the last item declared in a record, therefore the following reordering statements in the Subschema DDL are forbidden:

Examples:

In the Schema DDL:

```
01      A   TYPE SIGNED BINARY 31.
01      B   TYPE CHAR 20 DEPENDING ON A.
```

In the Subschema DDL:

```
01      B   TYPE CHAR 20 DEPENDING ON A.
01      A   TYPE SIGNED BINARY 31.       <===FORBIDDEN
```

In the Schema DDL:

```
01       A   TYPE CHAR 2.
  02     B   TYPE SIGNED BINARY 15.
  02     C.  OCCURS 10 DEPENDING ON B.
    03   D   TYPE CHAR 7.
    03   E   TYPE SIGNED BINARY 31.
```

In the Subschema DDL:

```
01       A   TYPE CHAR 2.
  02     C.  OCCURS 10 DEPENDING ON B.
    03   D   TYPE CHAR 7.
    03   E   TYPE SIGNED BINARY 31.
  02     B   TYPE SIGNED BINARY 15.       <===FORBIDDEN
```

4.2.6 Redefinition of Set Selection

A SET SELECTION Clause can always be redefined, but only BY APPLICATION. As we have seen, this redefinition is mandatory each time a set which is used in a SET SELECTION Clause is excluded. This redefinition can be specified on any other SET SELECTION.

4.3 WRITING THE SUBSCHEMA

The way the subschema is written will determine the effectiveness of the operation of the database. When writing the subschema, one must seriously take into account the following:

- database performance
- the effects which the definition of the subschema will have on the execution of data manipulation verbs

The following subsections discuss how database performance is affected by the way the subschema is defined. The effects which the definition of the subschema will have on the execution of DML verbs is also discussed.

4.3.1 Database Performance

Each time there is a change in the mapping of a record in the database (a change made in the Schema DDL) and consequently a change in the subschema, the DBCS must perform mapping conversions, which is a time-consuming operation for the CPU (see Figure 4-2). A new mapping can be the result of one of the following mechanisms, which were introduced in the beginning of this section:

- the exclusion of a field
- the reordering of field(s)
- transformations of field type(s)

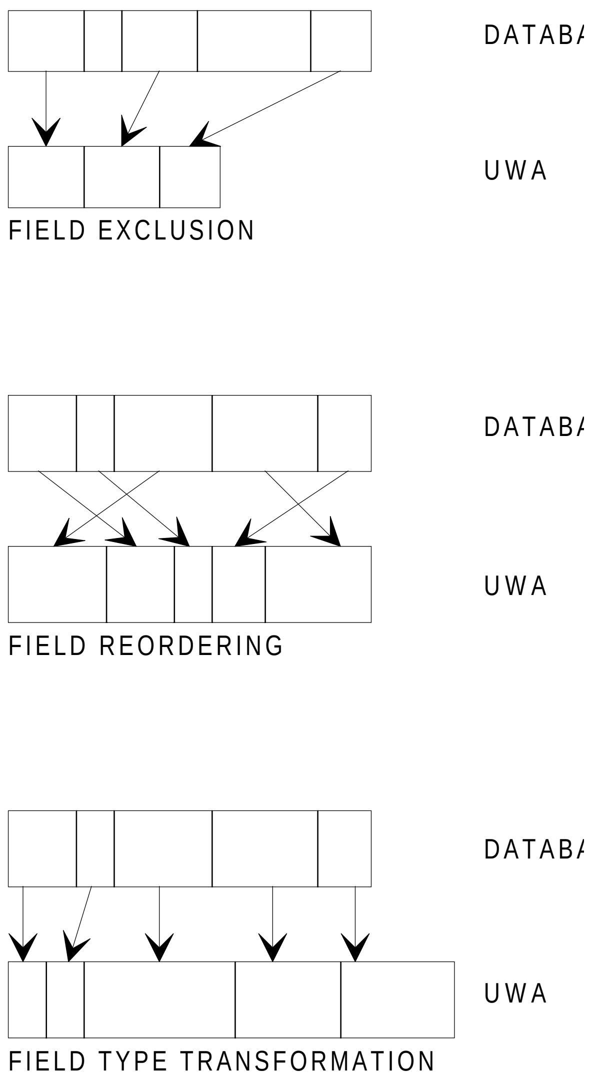


Figure 4-2. Different Mapping Between the UWA and the Database

To use an example, during the execution of a GET statement, instead of performing a simple move of the entire data part record from the database to the UWA, the DBCS must perform multiple moves with corresponding conversions (one for each record field).

4.3.2 Effects on Data Manipulation Verbs

Consider the effect which the following Schema DDL would have on the execution of data manipulation verbs:

```
                  SCHEMA DDL
AREA NAME IS A01.

AREA NAME IS A02.

AREA NAME IS A03.

RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA
    AREA-ID IS AREA-PARAM-1.
01  R01F01  TYPE CHAR 3.
01  R01F02  TYPE SIGNED BINARY 15.
```

And the following Subschema DDL:

```
                  SUBSCHEMA DDL

AREA   NAME IS A01.
                          (exclusion of area A02)
AREA   NAME IS A03.

RECORD NAME IS R01.
```

In light of the DDL and SDDL shown above, area A02 is excluded and therefore it is **impossible**, according to the defined database structure, to see record R01 in this area.

However, in terms of the following Subschema DDL, record R01 is not visible, but this is because of a **restriction** which is written into the SDDL:

```
               SUBSCHEMA DDL

AREA   NAME IS A01.

AREA   NAME IS A02.

AREA   NAME IS A03.

RECORD NAME IS R01
       WITHIN  AREA   A01 A03.
```

In this case, area AO2 is included but record RO1 is not visible from this area because of a restriction defined in the SDDL. If a field is excluded, then the DML STORE function is impossible on that record unless the missing field (whether it is a control field or not) has a DEFAULT VALUE declared in the Schema DDL.

In the subsections which follow, we will examine the effects which the definition of the subschema has on:

- retrieval verbs
- update verbs

4.3.3 Effects on Retrieval Verbs

In terms of the effects that the definition of the subschema has on DML retrieval verbs, one general rule must be remembered:

When an object which is not included in the subschema is encountered during the execution of a retrieval verb in a search domain, this object (record or area) is skipped (see Figure 4-3).

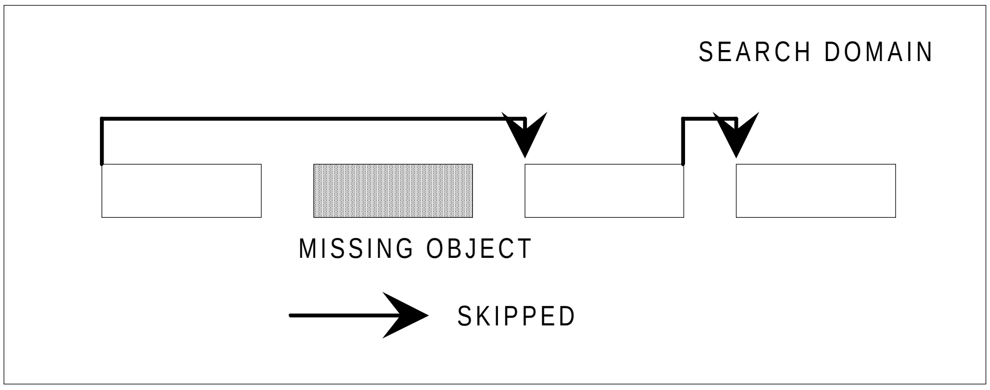


Figure 4-3. Excluded Objects are skipped by retrieval verbs

4.3.3.1 The Search Domain as a Set Occurrence

If there are excluded records and if the search domain is a set occurrence, these records are skipped during the execution of these two statements:

```
FIND ... WITHIN SET statement
```

```
ACCEPT ... FROM SET.statement.
```

See Figure 4-4 below.

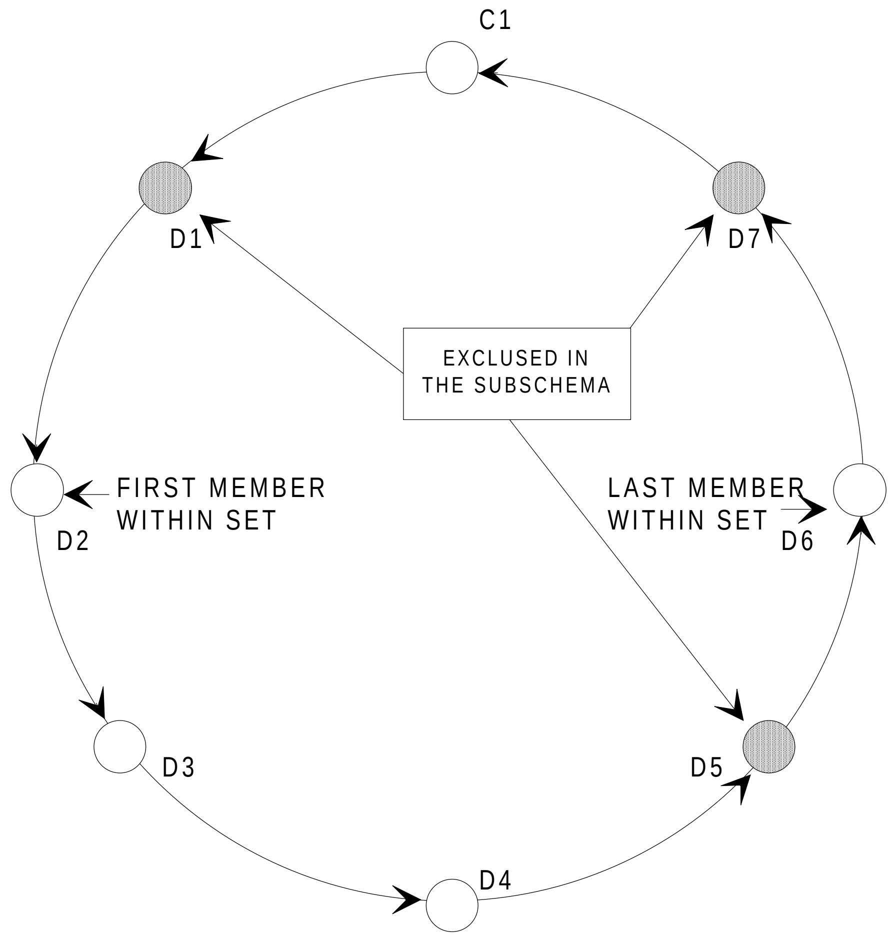


Figure 4-4. Members Excluded from a Set

In the set occurrence shown in Figure 4-4, members D1, D5 and D7 are not visible, therefore the following is true:

- the first member of the set is D2.
- the last member of the set is D6.
- the next member of D4 within this set is D6 (because D5 is skipped)

When multi-area sets are involved, a member can be a record which is restricted to a given number of areas. For example, in Figure 4-5, members D1, D2, D6 and D7 are not visible from AREA-2. Therefore:

- the first member of the set is D3
- the last member of the set is D5

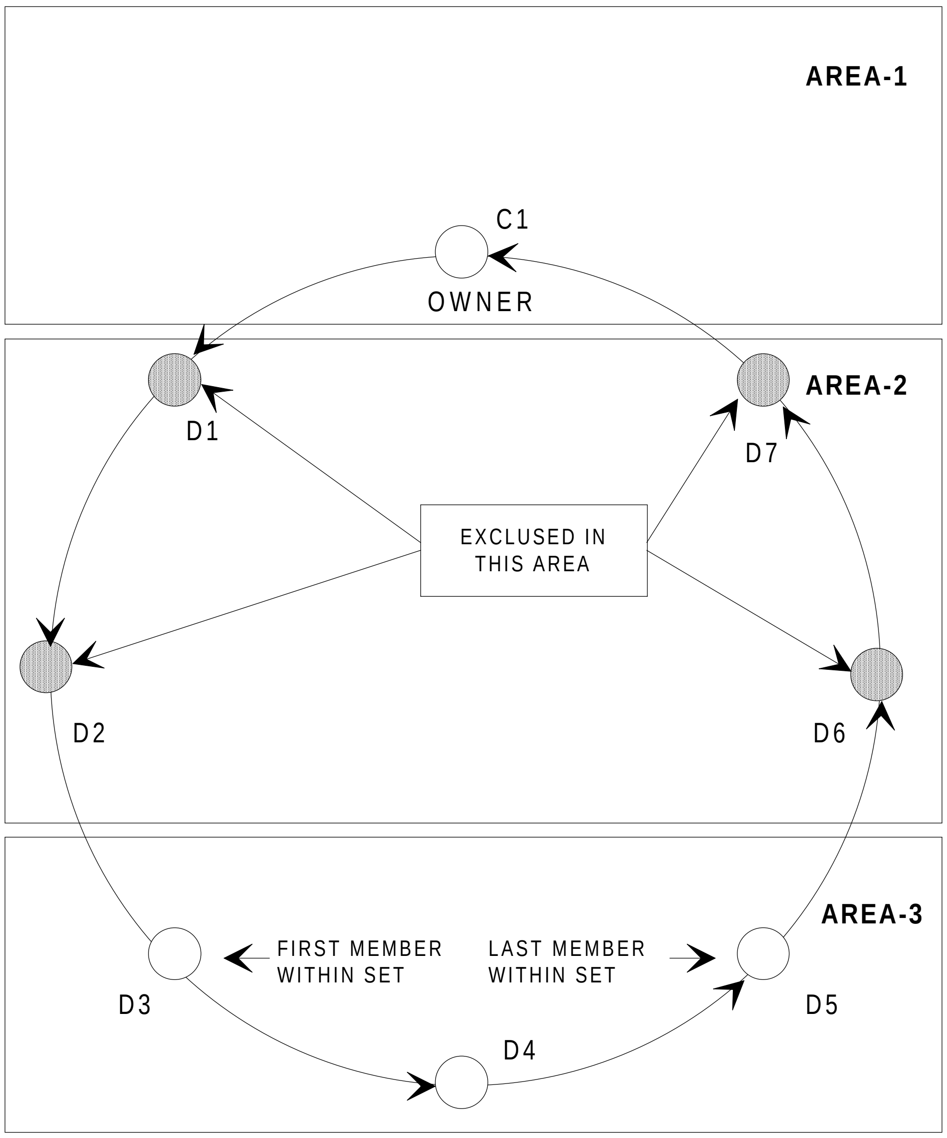


Figure 4-5. Members Excluded from an Area

Consider a case in terms of the preceding occurrence diagram where AREA-2 is excluded from the subschema (see Figure 4-6). The following becomes true:

- the first member is still D3
- the last member is still D5

However, these members are no longer accessible by the owner of the set occurrence because the path from C1 to D3 goes through AREA-2, which is an unknown area; in other words, an area which was not readied. In such a case, statements such as FIND FIRST/LAST WITHIN SET lead to a database exception at run-time.

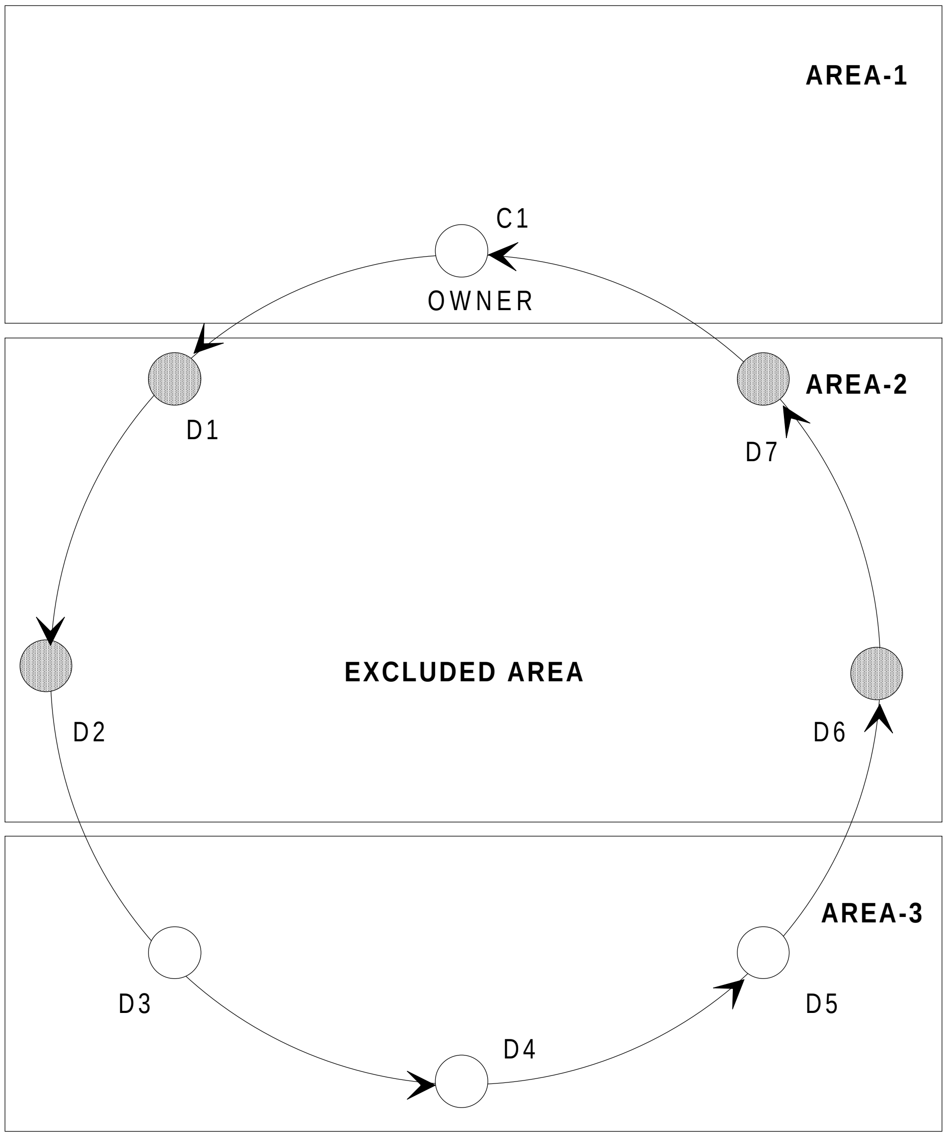


Figure 4-6. Missing Members Within A Set: an Excluded Area

4.3.3.2 The Search Domain as an Area

If there are excluded records when the search domain is an area, these records are skipped during the execution of the following statements:

```
- FIND  ...  WITHIN AREA statement
- ACCEPT ... FROM AREA current statement.
```

See Figure 4-7.

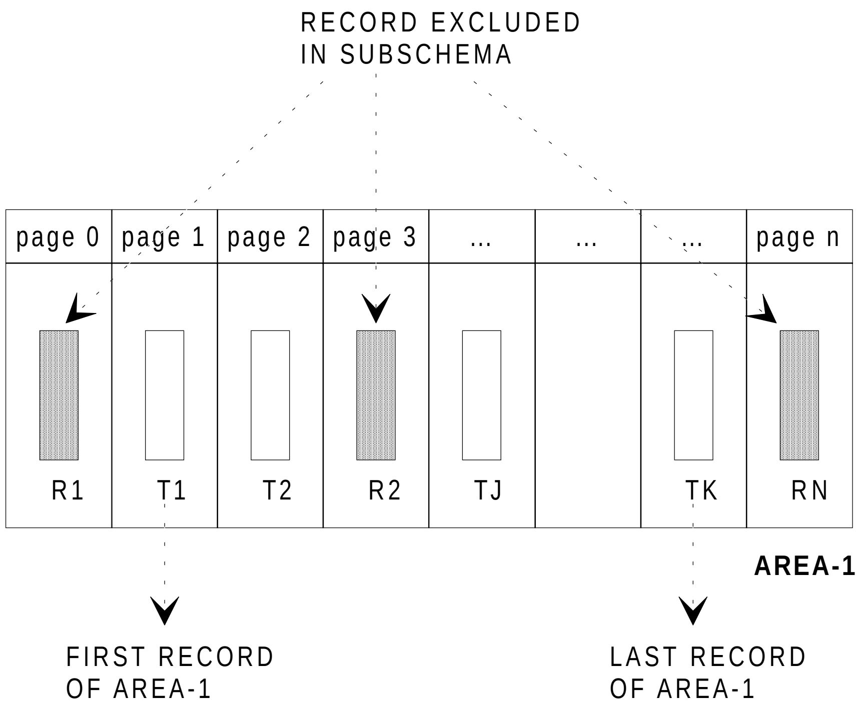


Figure 4-7. Excluded Records Within An Area

In this area, records R1, R2 and RN are not visible, therefore, the following statements are true:

- the first record of the area is T1
- the last record of the area is TK
- the next record after T2 within this area is the TJ (R2 is skipped)

4.3.3.3 The Search Domain as a Secondary Key

If the search domain is a secondary key and there are missing records, they are skipped during the execution of the following statements:

```
- FIND   ... WITHIN KEY statements
- ACCEPT ... FROM KEY statements
```

See Figure 4-8.

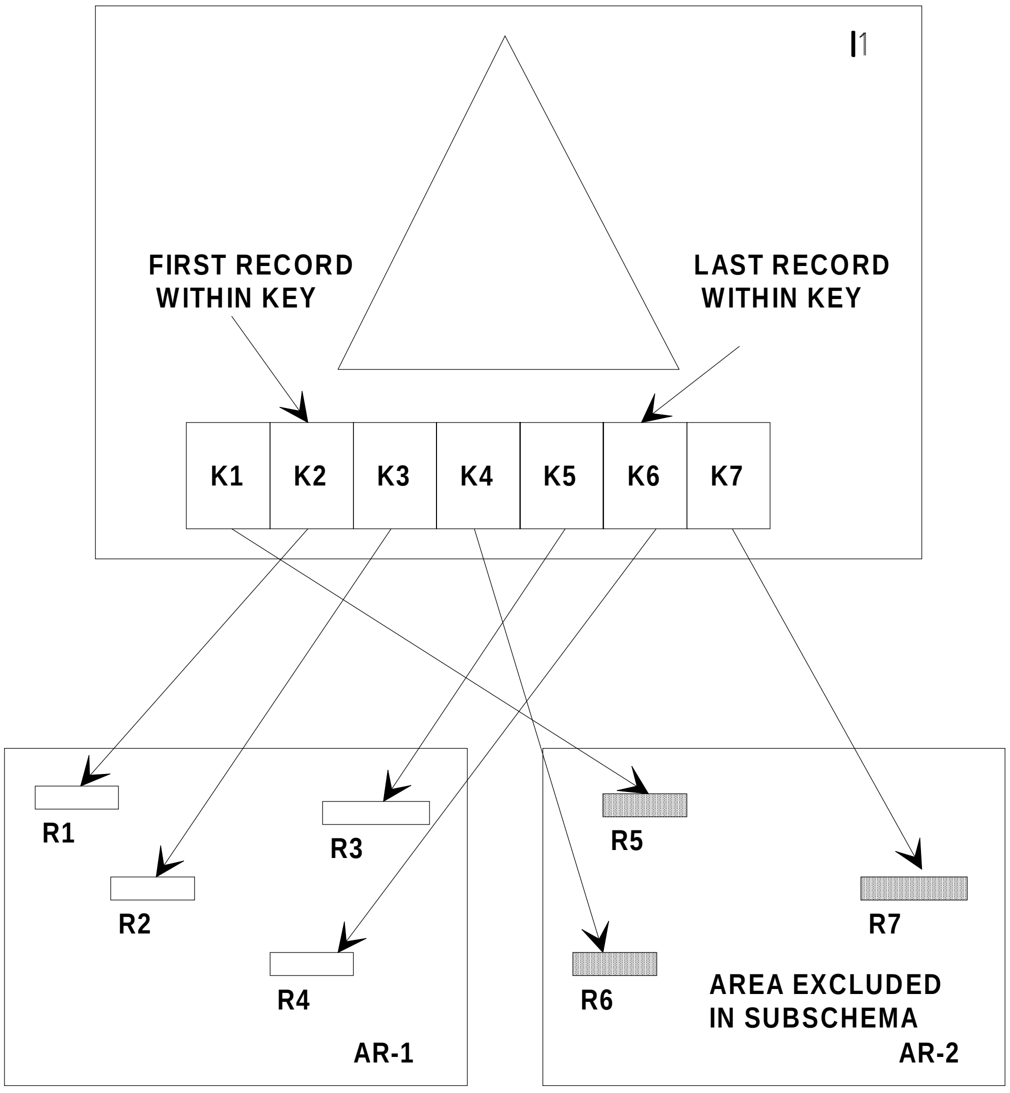


Figure 4-8. Missing Records Within An Area: the Case of an Index

In this case, area AR-2 is excluded and therefore records R5, R0 and R7 are not visible. The following holds true:

- the first record within this key (key K2) is R1
- the last record is R4 (key K6)
- the next record of R2 (key K3) within this key is record R3 (R6 is skipped)

4.3.4 Effects on Update Verbs

In terms of the effect which the definition of the subschema has on DML update verbs, one general rule must be remembered:

When an object which is not included in the subschema is encountered during the execution of an DML update verb, the update statement is denied and a database exception occurs.

See Figure 4-9, below.

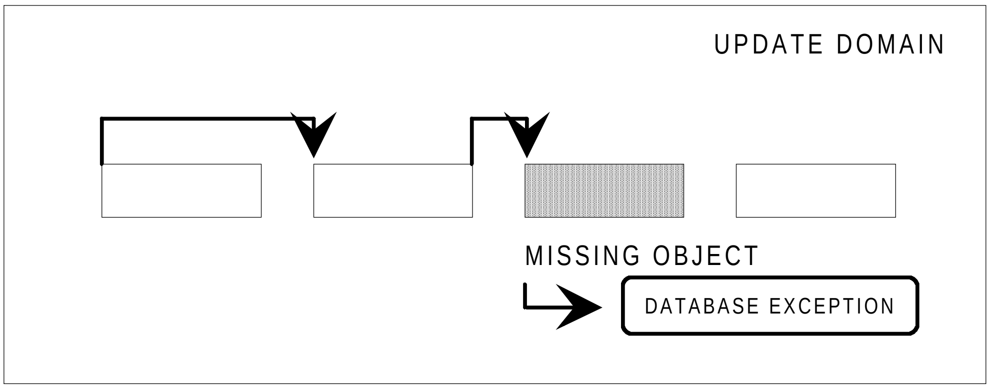


Figure 4-9. Database exception caused by missing objects for Update Verbs

4.3.4.1 Effects on the MODIFY MEMBERSHIP Statement

If a sort key field is missing in a record, the MODIFY MEMBERSHIP statement directed to an occurrence of this record is denied because this record cannot be inserted in its new set occurrence (see Figure 4-10).

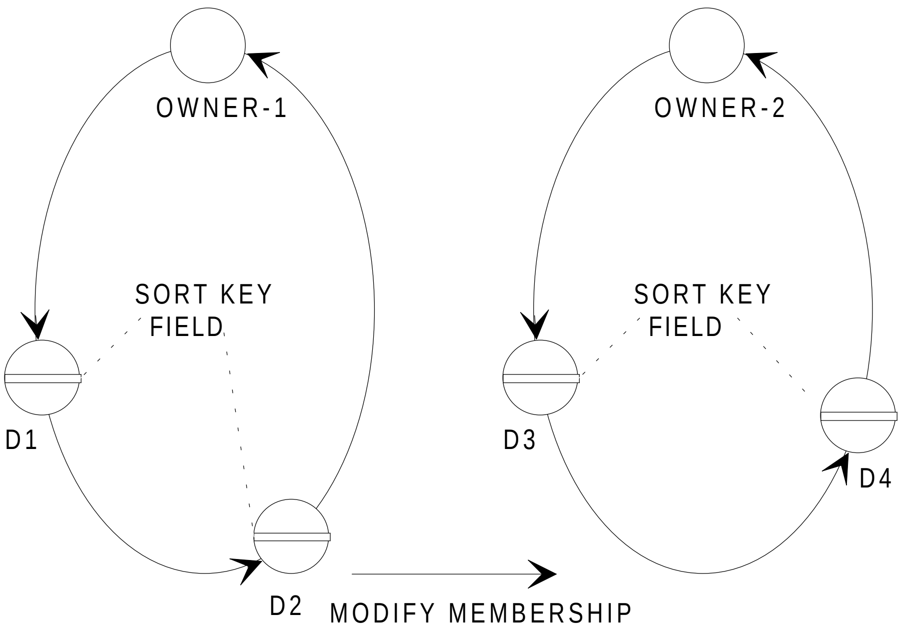


Figure 4-10. MODIFY MEMBERSHIP Statement on a Record with a Missing Sort Key Field

4.3.4.2 Effects on the ERASE Statement

During an ERASE ALL statement, if the DBCS encounters an area excluded from the subschema, the statement cannot be performed. In Figure 4-11, the ERASE ALL statement directed to record C1 will set out to delete record C1 and all its members: D1, D2, D3, E1 and E2.

But, since records E1 and E2 are in an excluded area (AREA-3), the statement is denied at run-time. This is the equivalent of the IDS/II-V3 error *AREA NOT OPENED*.

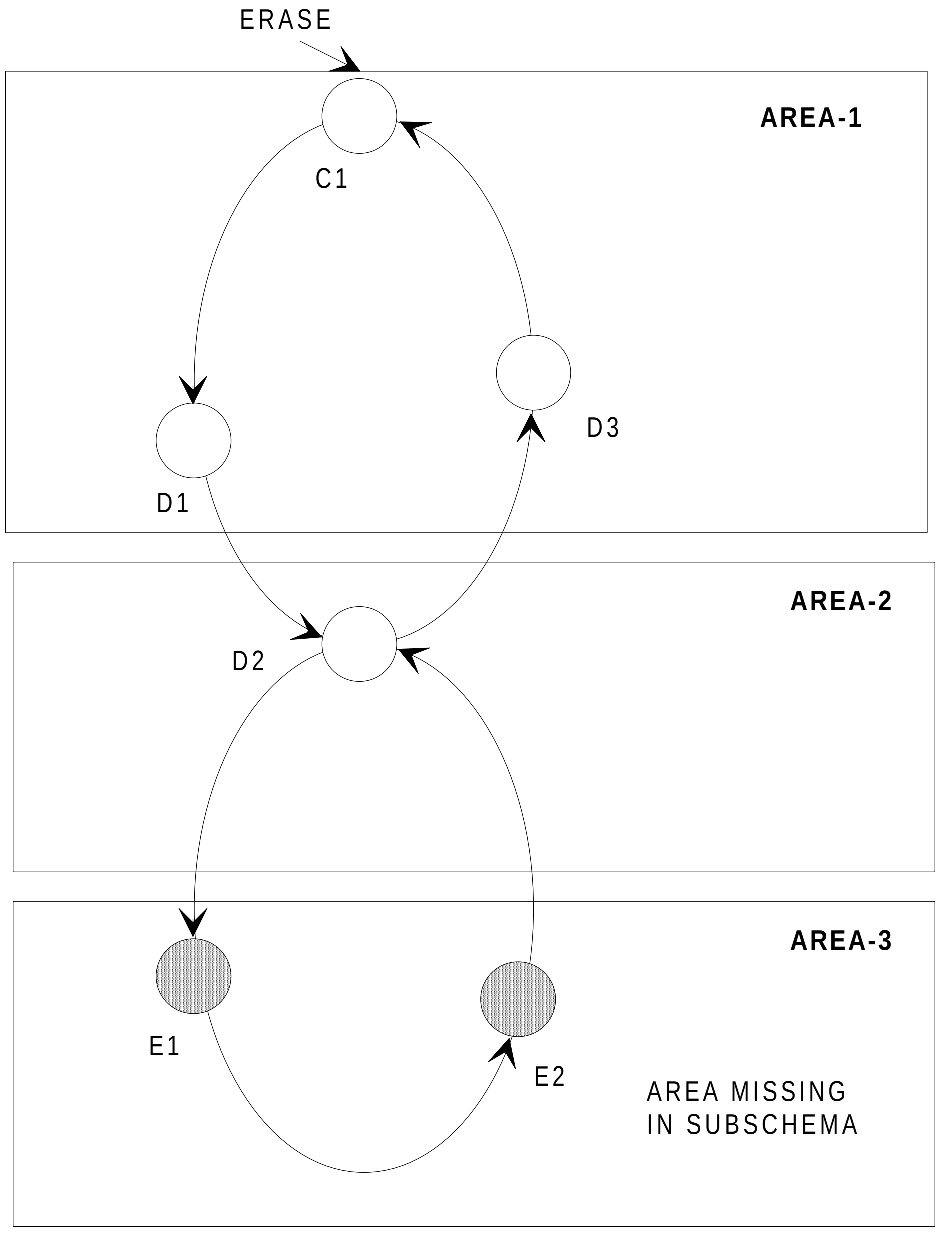


Figure 4-11. ERASE ALL Statement directed to a Set with an Excluded Area

5. Data Manipulation Retrieval Functions

This section describes the four data manipulation functions which retrieve information from the database.

1) The FIND function locates a record in the database. Record selection is specified by two elements:

   - the **ordered class** which is to be searched: database, area, secondary key or set occurrence
   - and the **selection criteria** within this class: position, record-type, field contents

2) The GET function moves the contents of all or part of a located record from the database to the User Working Area.

3) The ACCEPT function serves a number of purposes:

   - it makes available to the program the data-base-keys contained in the currency indicators
   - it retrieves the realm-name which is coded in a data-base-key
   - it makes available to the program the data-base-keys of the NEXT, PRIOR or OWNER of a record within a set
   - it retrieves DMCL information from the object schema
   - it makes available to the program the data-base-keys of the NEXT or PRIOR of a record within a secondary key.

4) The Data Base Condition function ("IF") may test the status of a record with respect to one or several set occurrences, or the status of a set occurrence.

A detailed explanation of each of these functions is contained in this section.

5.1 FIND

5.1.1 General

The FIND function **locates a record in the database**.

However, the contents of the record are not made available in the User Working Area (UWA) until a GET is performed. The reason for the distinction between these two functions is that several successive FIND operations may be necessary to locate a record, which is followed by only one GET.

In IDS/II, retrieval methods fall into nine categories :

1) direct access within the database based on the data-base-key
2) direct access within the database based on the CALC-key
3) direct access within the database or within the area (depending on the scope of the key) based on the secondary key
4) sequential access within an area based on a relative position
5) sequential access to a member within a set occurrence based on a relative position
6) sequential access to a member within a set occurrence based on field contents,
7) sequential access to a record within a secondary key based on a relative position
8) sequential access to a record within a secondary key based on key components
9) access to the owner within a set occurrence

Each of these retrieval methods is explained in detail in the sections directly following.

After the successful completion of a FIND statement, the record retrieved becomes the **current-of-run-unit**; its record-name and area-name are put into DB-RECORD-NAME and DB-REALM-NAME respectively. Unless corresponding currencies are specified in the RETAINING Clause, the record also becomes:

- the current of its record-type
- the current of its realm
- the current-of-set-type of all the sets of which it is the owner or a currently connected member
- and the current-of-key-type of all the keys which are declared for this record.

5.1.2 Direct Access within the Database, using the Data-base-key

This type of access requires only one page transfer. The data-base-key identifying the record to be retrieved is either supplied in the UWA or contained in a currency indicator.

5.1.2.1 Data-base-key Supplied in the UWA

The statement :

```
    FIND [record-name] DB-KEY is identifier.
```

applies to any record, whatever its LOCATION MODE. The parameter "identifier" is declared in the User Working Area with USAGE IS DB-KEY and must be initialized with a data-base-key. The area code part of the data-base-key allows the selection of the area. The area-key part of the data-base-key gives the page number within the area and the line number within the page (see Figure 5-1).

If "record-name" is specified in the statement, the DBCS will check that the record found belongs to this record-type.

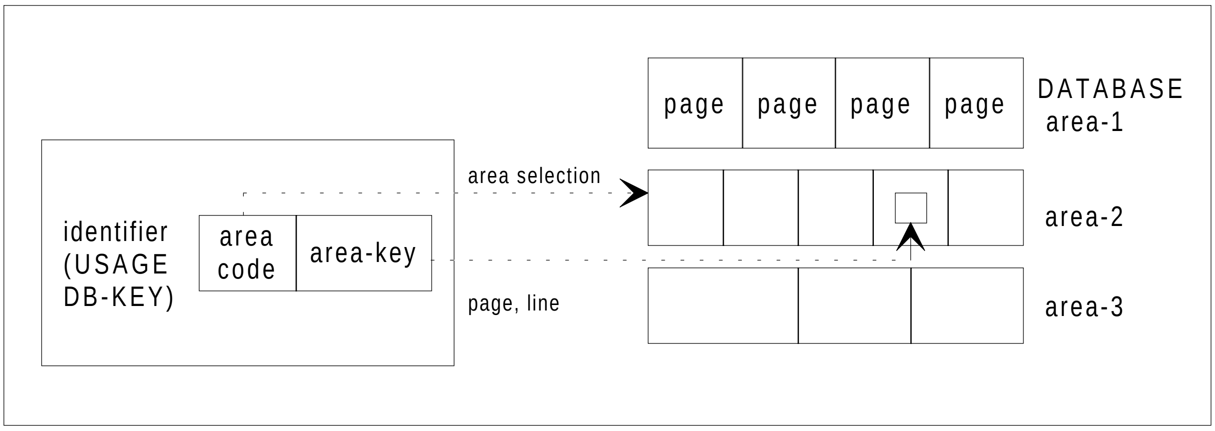


Figure 5-1. Direct Access by Data-base-key

The data-base-key supplied in the UWA can be either one of the following :

- the result of a user-specified algorithm. This is usually the case with records whose LOCATION mode is DIRECT, the same data-base-key computation being performed for the storage and the retrieval of the record.
- the result of a preceding save of a currency indicator. For example, after a STORE (whatever the LOCATION MODE), the data-base-key assigned to the record can be obtained in a UWA data item (declared with USAGE IS DB-KEY) by an ACCEPT statement, as in the example below:

```
STORE REC-09
ACCEPT SAVE-DBK FROM CURRENCY
```

Later in the processing, the record can be retrieved directly by the statement:

```
        FIND REC-09 DB-KEY IS SAVE-DBK
```

5.1.2.2 Data-base-key Contained in a Currency Indicator

When the programmer is sure that a currency indicator still points to the record which is to be retrieved, the FIND CURRENT statements below can provide him with the same direct access as the "FIND DB-KEY IS identifier" statement:

```
    FIND CURRENT [record-name] WITHIN key-name.
    FIND CURRENT [record-name] WITHIN realm-name.
    FIND CURRENT [record-name] WITHIN set-name.
    FIND CURRENT record-name.
    FIND CURRENT.
```

To clarify the functional similarity between these two statements, the parameter "identifier" from the "FIND DB-KEY IS identifier" statement is replaced by the corresponding **currency indicator** retrieved by the FIND CURRENT statement:

- current-of-key
- current-of-realm
- current-of-set-type
- current-of-record-type
- current-of-run-unit

The last FIND CURRENT statement listed above allows the user to update currencies previously retained during the retrieval of the **current-of-run-unit**.

If "record-name" is specified in the first three statements, the DBCS will check that the current record belongs to this record-type.

5.1.3 Direct Access within the Database using the CALC-key

This type of access requires an average number of page transfers close to 1.

5.1.3.1 Unique Record or First Duplicate

The statement:

```
FIND ANY record-name.
```

applies to a record whose LOCATION MODE is CALC.

If DUPLICATES ARE NOT ALLOWED, the FIND function will retrieve the unique record, if any, having the required CALC-key value. If DUPLICATES ARE ALLOWED, the FIND function will retrieve the first, if any, of the records having the required CALC-key value.

It is necessary for the programmer to have initialized the CALC fields in the record of the UWA; if the record can be located in more than one area, the AREA-ID field required for the area selection also needs to have been initialized.

To determine the "bucket" in which the record is placed, the DBCS executes the same "CALC-key-transformation" algorithm as for the storage of the record. If there is a chain of synonyms in that bucket, the DBCS searches it for the first record of that type whose CALC-key value is equal to that of the UWA (see Figure 5-2).

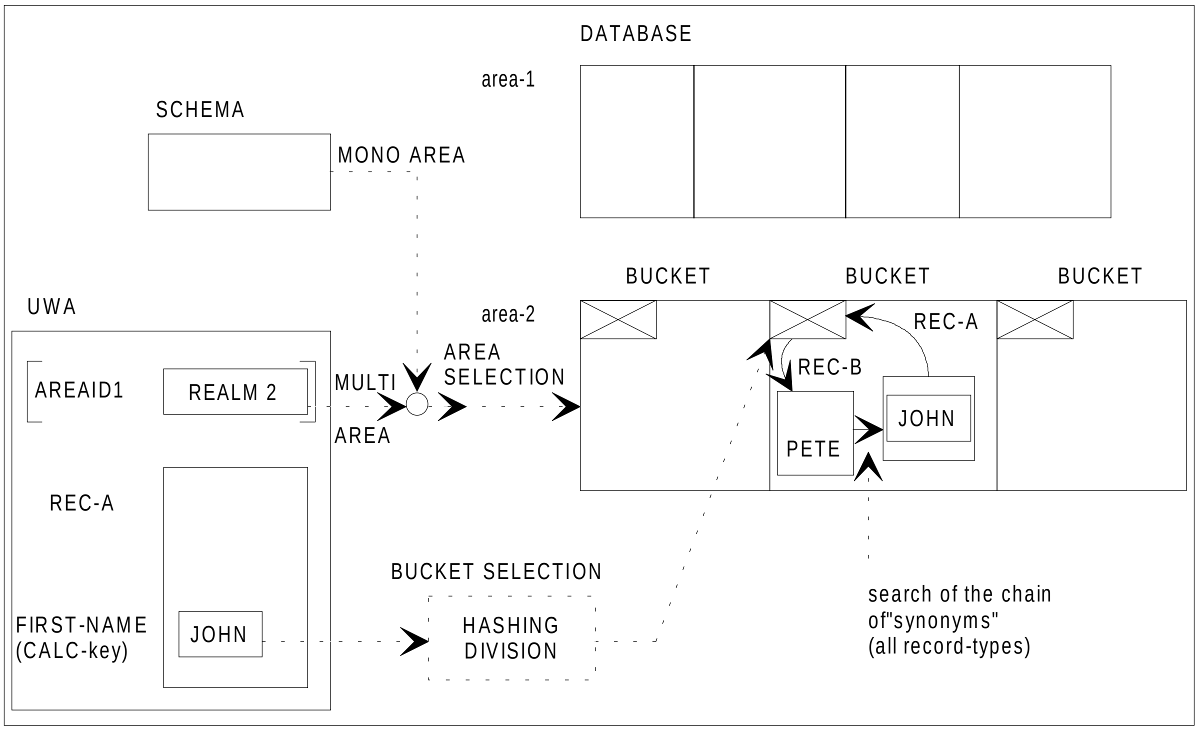


Figure 5-2. Direct Access by CALC-key

5.1.3.2 Other Duplicates

The following statement:

```
    FIND DUPLICATE record-name.
```

applies to a record whose LOCATION MODE IS CALC with the further condition that DUPLICATES ARE ALLOWED. This means that two records of the same type and having the same CALC-key value may be stored in the same area. Since these two records are synonyms, they randomize to the same bucket and participate in the same CALC chain; The statement "FIND ANY record-name" retrieves the first DUPLICATE in the CALC chain; the statement "FIND DUPLICATE record-name" retrieves the following DUPLICATES. The relative position of the DUPLICATES in the CALC chain is not related to their chronological insertion and cannot be used functionally.

The "FIND DUPLICATE record-name" statement assumes that the current-of-run-unit is already one of the DUPLICATES. The DBCS searches the CALC chain, beginning with the current record of the run-unit, for the next record of the same type whose CALC-key value is equal to that of the current-of-run-unit (see Figure 5-3). The corresponding field in the UWA is not used for comparison.

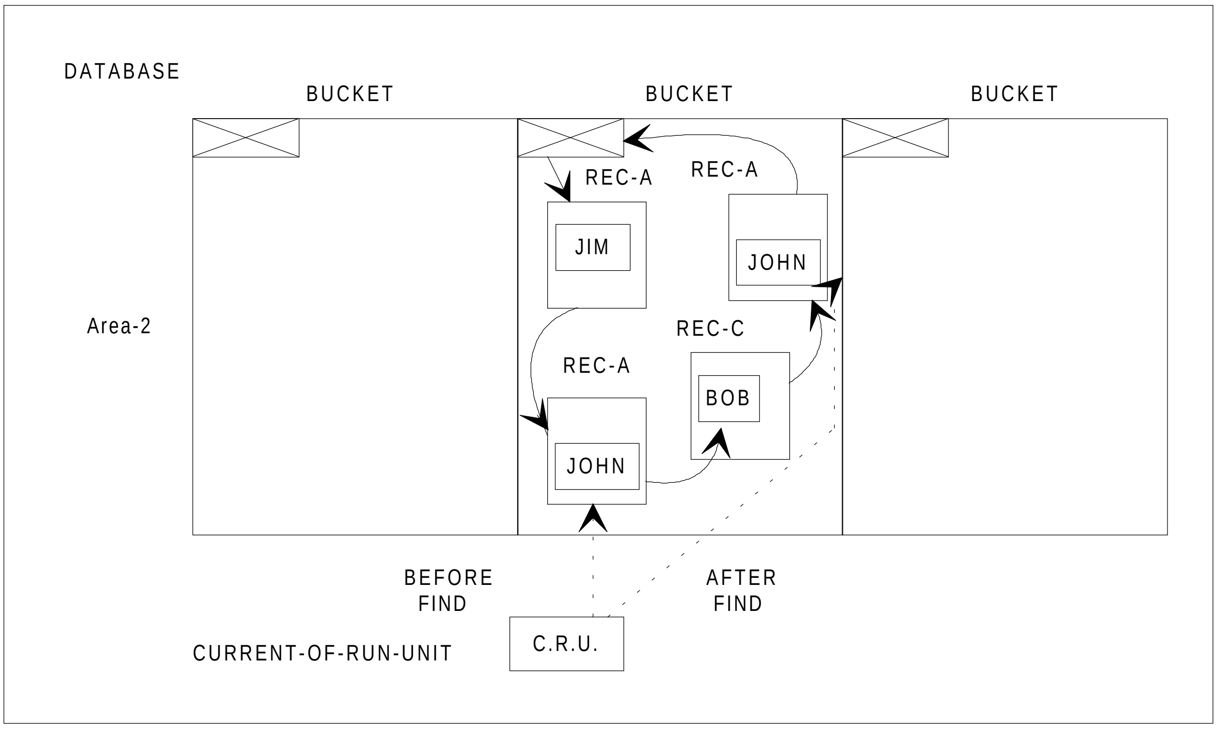


Figure 5-3. Retrieval of CALC Duplicates

Though it does not actually provide a direct access to a record, the "FIND DUPLICATE record-name" statement is documented in this section because of its close connection with the "FIND ANY record-name" statement.

5.1.4 Direct Access using the Secondary Key

5.1.4.1 Unique Key or First Duplicate

The same rules of logic apply to the execution of the following statement :

```
FIND ANY [record-name] USING key-name.
```

as those which apply to direct access using CALC-KEY statements, which were explained earlier in this section. These rules are given below:

1) If DUPLICATES ARE NOT ALLOWED is specified, the FIND function will retrieve the unique record, if any, having the corresponding secondary key value.

2) If DUPLICATES ARE ALLOWED is specified, the FIND function will retrieve the first, if any, of the records having the corresponding secondary value.

It is necessary for the programmer to have initialized the fields of the key in the record of the UWA for a mono-record-type key, or to have initialized the db-parameters for a multi-record-type key. If the scope of the key is the area, the AREA-ID, which is defined in the KEY Entry, is required for the area selection.

Examples:

Consider, for example, the Schema DDL below, which defines a mono-record-type key:

```
RECORD NAME IS EMPLOYEE
KEY NAME IS K01 USING ASCENDING NAME.
       ...
01     NAME    TYPE IS CHAR 20.
01     NUMBER  TYPE IS SIGNED BINARY 31.
01     ADDRESS TYPE IS CHAR 30.


       ...

KEY NAME IS K01
    DUPLICATES ARE NOT ALLOWED.
```

The following statements retrieve the employee whose name is "SMITH":

```
MOVE "SMITH" TO NAME.

FIND ANY EMPLOYEE USING K01.
```

Now, consider the the Schema DDL below, which defines a multi-record-type key:

```
RECORD NAME IS MAN
KEY NAME IS K01 USING NAME.
       ...
01     NAME    TYPE IS CHAR 20.
01     NUMBER  TYPE IS SIGNED BINARY 31.
01     ADDRESS TYPE IS CHAR 30.



RECORD NAME IS WOMAN
KEY NAME IS K01 USING NAME.
       ...
01     NAME    TYPE IS CHAR 20.
01     NUMBER  TYPE IS SIGNED BINARY 31.
01     ADDRESS TYPE IS CHAR 30.


       ...

KEY NAME IS K01
    USING      EMPLOYEE-NAME
    DUPLICATES ARE ALLOWED.
```

The following statements retrieve the male or female employee whose name is "SMITH":

```
    MOVE "SMITH" TO EMPLOYEE-NAME.

    FIND ANY  USING K01.
```

If the search is restricted to female employees, the record-type must be specified as shown below:

```
    MOVE "SMITH" TO EMPLOYEE-NAME.

    FIND ANY  WOMAN USING K01.
```

5.1.4.2 Other Duplicates

The statement:

<u>FIND</u> <u>DUPLICATE</u> [record-name] <u>USING</u> key-name.

applies to a record with the condition that DUPLICATES ARE ALLOWED. For a multi-record key, this statement retrieves a record of the same record-type when "record-name" is specified.

If the scope of the key is the area, the statement "FIND DUPLICATE USING key-name" retrieves a record only in the area in which the prior record was found.

The FIND function will retrieve the next record whose key value is equal to that of the current-of-key-name. With the "FIND duplicate record-name" statement, the corresponding field in the UWA is not used for a comparison. The key value used for the comparison is the key of the record itself, in the database.

5.1.5 Sequential Access within an Area using a Relative Position

The record to be retrieved is defined by its **relative position in an area** regardless of its participation in any set. The order which defines the relative position of records in an area is that of increasing area-keys. The position may be relative to either :

- the ends of the area (FIRST, LAST, positive or negative ordinal)
- or the current record of the area (NEXT, PRIOR)

Since an area may contain several types of records, it is possible to specify that the relative position is to be interpreted either **independently** of the record-type or **within** a given record-type. In the latter case, the records of a different record-type are ignored.

If the record sought is not found when the end of an area is encountered, an end-of-realm condition is returned in DB-STATUS. Currencies are left in the state that they were in before the FIND statement was invoked.

5.1.5.1 Using a Position Relative to the Ends of the Area

In the area specified by **realm-name**, the statements a) or b) below locate either:

- the first record of any type (with the lowest area-key)
- the last record of any type (with the highest area-key)
- or the type specified by **record-name**

```
a) FIND FIRST [record-name] WITHIN realm-name
b) FIND LAST  [record-name] WITHIN realm-name
```

The statements:

```
FIND integer      [record-name] WITHIN realm-name
FIND identifier   [record-name] WITHIN realm-name
```

locate the **nth** record of any type (or of the type indicated by record-name) in the area specified by "realm-name"**, n** represents a positive or negative integer defined in the first format of the statement (FIND integer format) or else defined in a numeric field of the UWA ("identifier" of second format).

If **n** is positive, the search is performed in the forward direction starting from the lowest area-key. If **n** is negative, the search is performed in the backward direction starting from the highest area-key.

5.1.5.2 Using a Position Relative to the Current-of-area

The statements:

```
    FIND NEXT  [record-name] WITHIN realm-name
    FIND PRIOR [record-name] WITHIN realm-name
```

locate the NEXT or PRIOR record of any type (or of the type specified by record-name) with respect to the current-of-realm of the area specified by realm-name.

Figure 5-4 illustrates the action of the preceding statements.

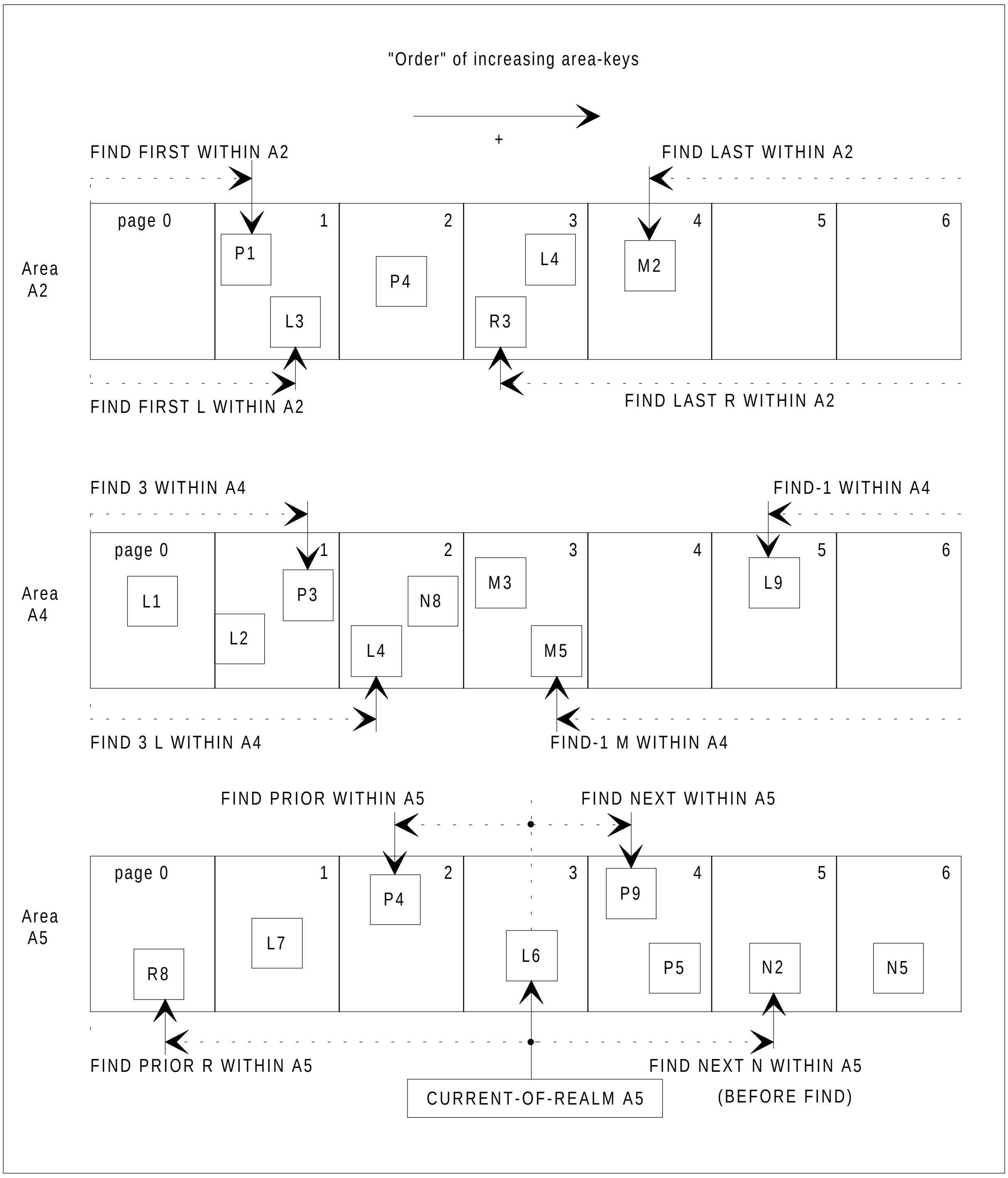


Figure 5-4. Sequential Access Within an Area Using a Relative Position

5.1.6 Sequential Access to a member within a Set Occurrence using a Relative Position

A set occurrence is identified by the current-of-set-type, which may be an owner or a member. Sets have a conventional order which establishes a distinction between a search in the forward (NEXT) direction and in the backward (PRIOR) direction.

The position of the member record within the set occurrence can be relative to either:

- the owner record (FIRST, LAST, positive or negative ordinal)
- the current record of the set (NEXT, PRIOR)

As in the case of an area, a choice can be made concerning the scope of a search within a set occurrence. Either :

- the search is performed on all member records, whatever their type (multi-member-type set)
- or the search is performed only on member records of a given type (multi or mono-member-type set). Records of another type are ignored.

If the record sought is not found when the owner of the set is encountered, the search terminates and an end-of-set condition is returned in DB-STATUS. The currencies are left in the state that they were in before the FIND statement was invoked.

5.1.6.1 Using a Position Relative to the Owner

The statements:

```
FIND FIRST  [record-name] WITHIN set-name
FIND LAST   [record-name] WITHIN set-name
```

locate, within the set occurrence specified by set-name, the following :

- the first member record (forward search originating from owner)
- or the last member record (backward search originating from owner) of any type or of the type specified by record-name. The set occurrence searched is dentified by its current-of-set-type.

Figure 5-5, further on in this section, illustrates the action of these statements

The following statements:

```
FIND integer    [record-name] WITHIN set-name
FIND identifier [record-name] WITHIN set-name
```

locate the **nth** member record of any type, or of the type specified by **record-name**, in the set occurrence specified by **set-name**. The set occurrence searched is identified by its current-of-set-type.

n is a positive or negative integer defined in the FIND integer statement (first format) or in a numeric field of the UWA ("identifier" of the second format). If **n** is positive, the search is performed in the forward direction starting from the owner. If **n** is negative, the search is performed in the backward direction, starting from the owner.

Figure 5-6 illustrates the action of these statements.

5.1.6.2 Using a Position Relative to the Current-of-set-type

The statements:

```
FIND NEXT  [record-name] WITHIN set-name
FIND PRIOR [record-name] WITHIN set-name
```

locate the NEXT or PRIOR member record of any type or of the type indicated by **record-name**. The search is performed with respect to the current-of-set-type of the set specified by **set-name.**

When the current-of-set-type is the owner of the set occurrence, the statements are equivalent to "FIND FIRST [record-name] WITHIN set-name" and "FIND LAST [record-name] WITHIN set-name" respectively.

Figure 5-5 illustrates the action of these statements.

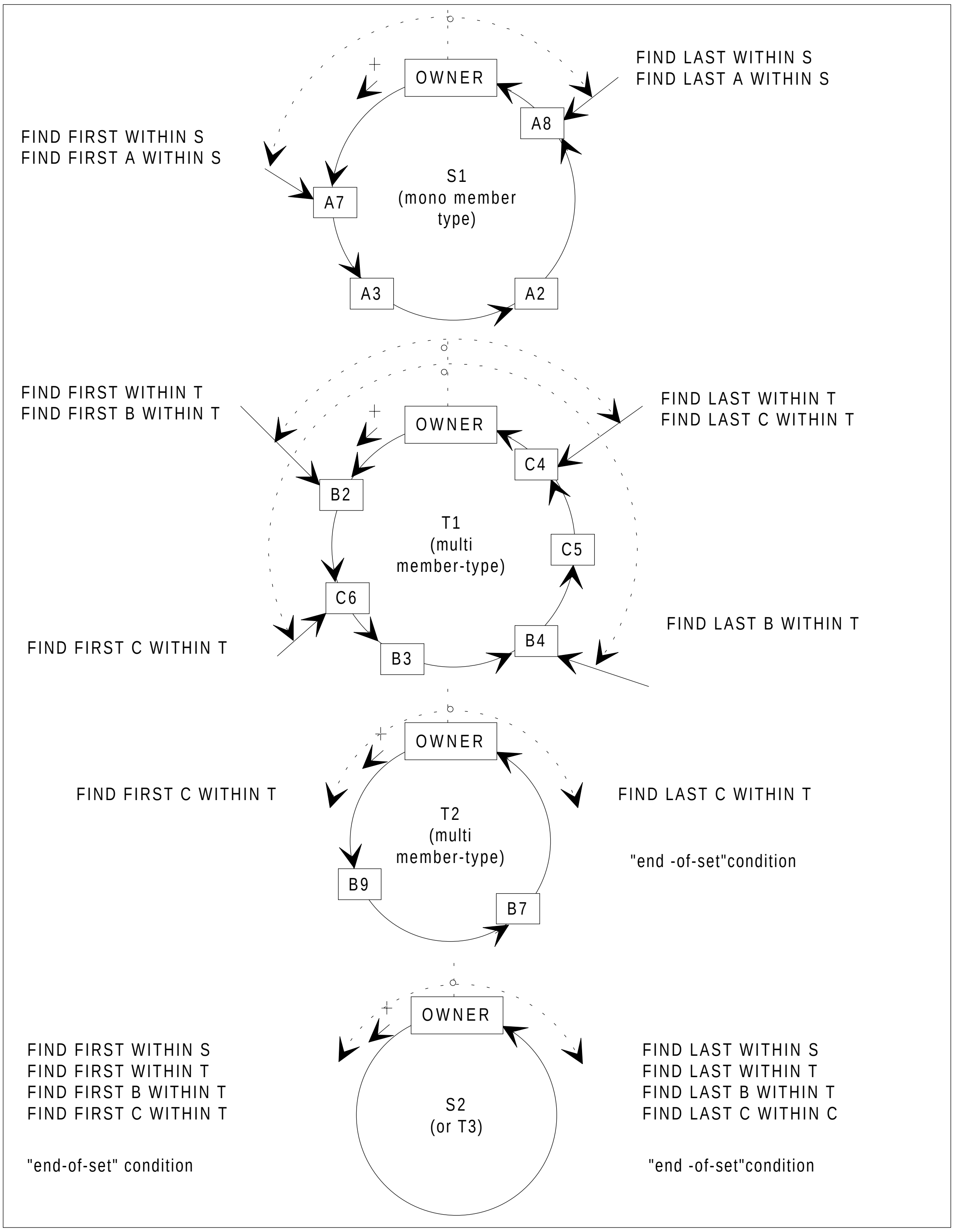


Figure 5-5. Sequential Access to a Member Within a Set Occurence (FIRST, LAST)

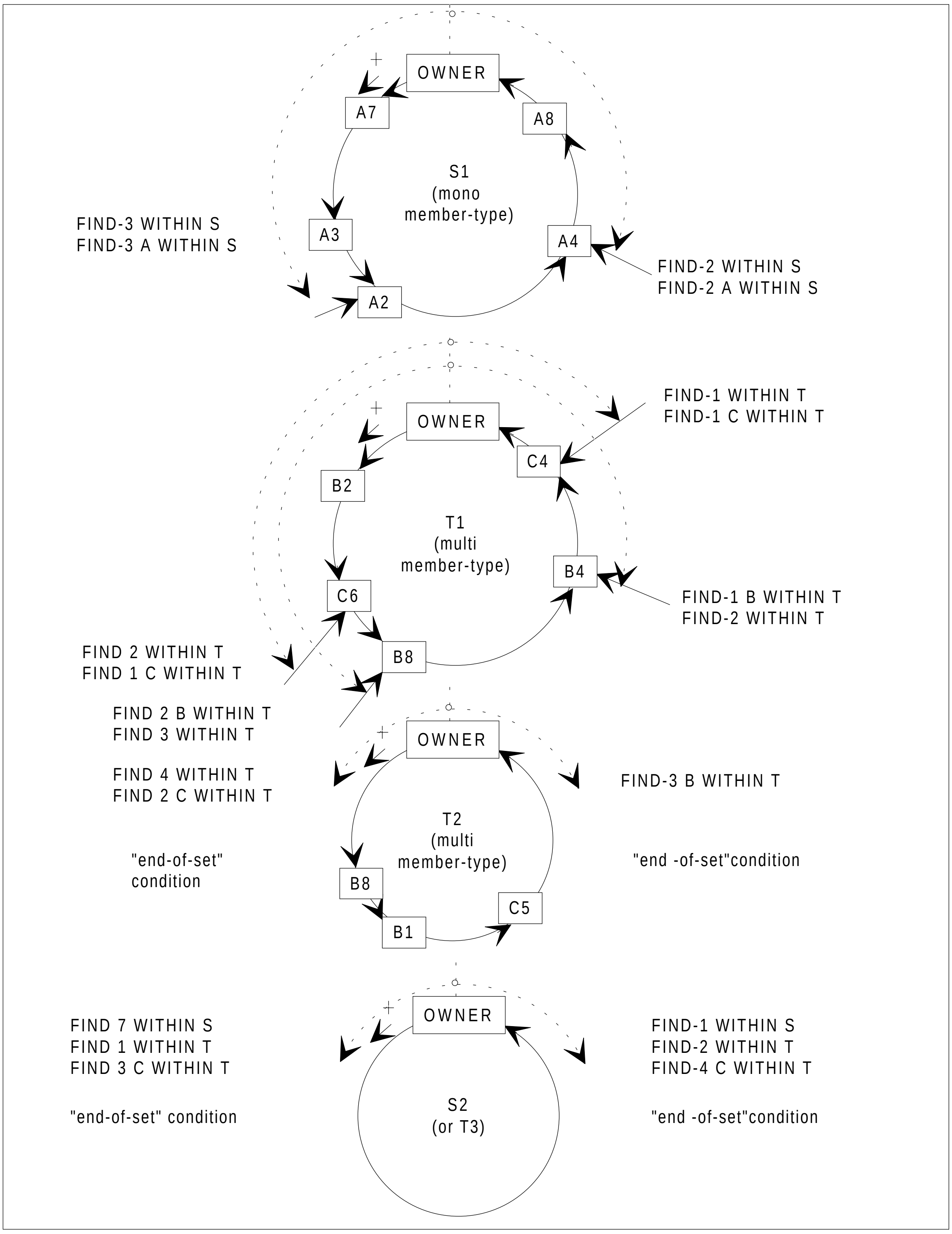


Figure 5-6. sequential Access to a Member Within a Set Occurence (ORDINAL)

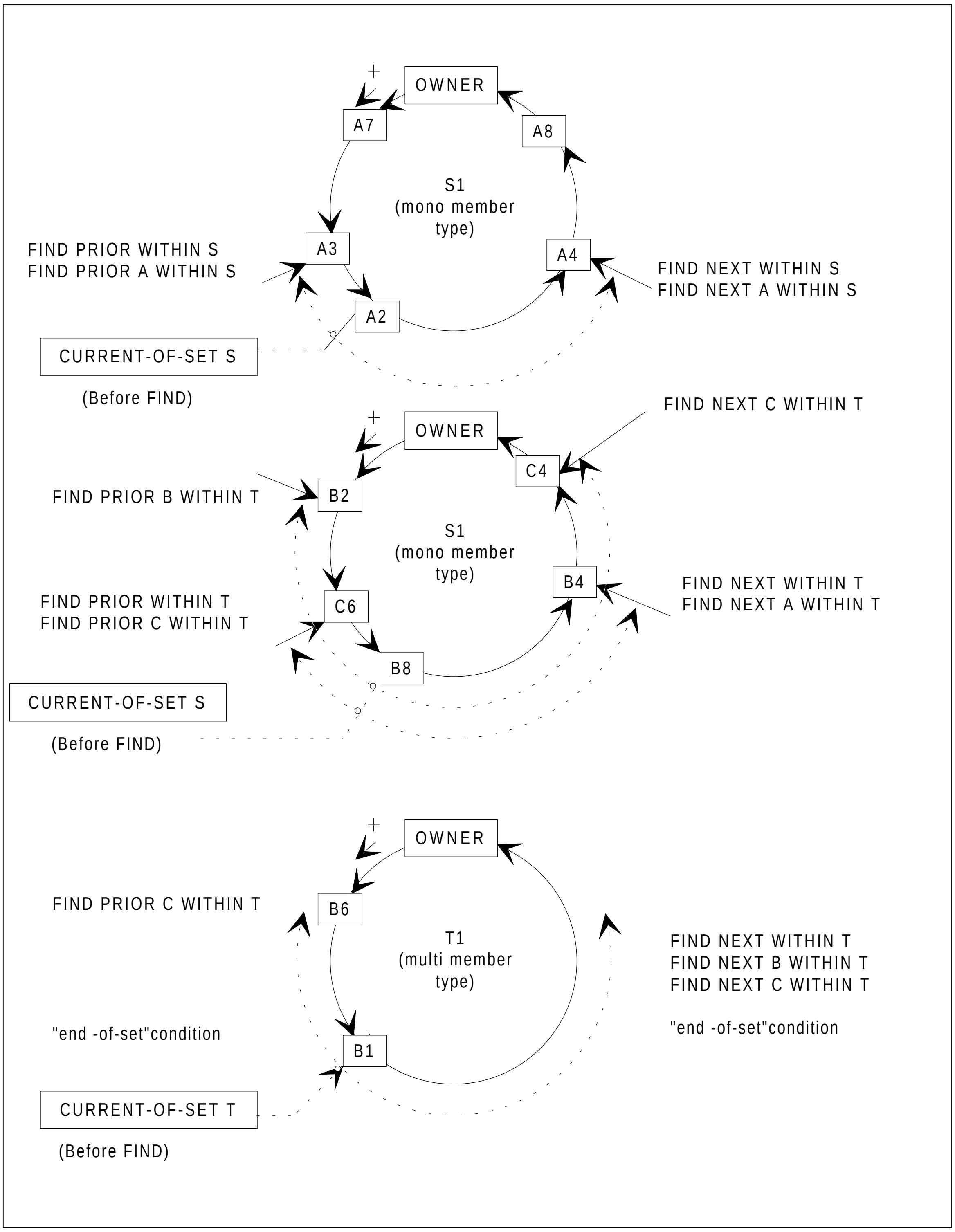


Figure 5-7. Sequential Access to a Member Within a Set Occurence (NEXT,PRIOR)

5.1.7 Sequential Access to a Member within a Set Occurrence using Field Contents

5.1.7.1 Search from the Owner

The following statement :

```
FIND record-name WITHIN set-name [CURRENT] [USING {identifer} ...]
```

locates the member record of the type specified by **record-name** in which the contents of the fields specified by the parameter "identifier" are equal to the contents of corresponding fields in the UWA.

The search proceeds from the owner in the forward direction. If the owner is encountered before the record is found, a **record not found** condition is returned in DB-STATUS.

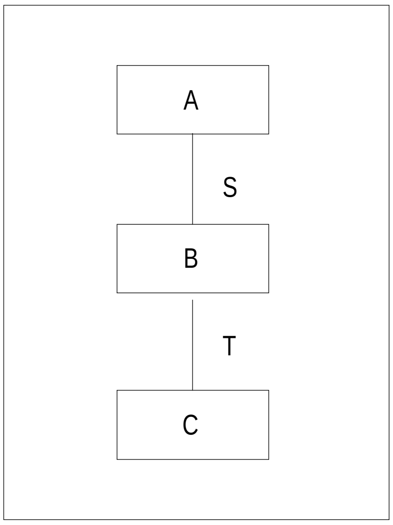
The keyword CURRENT, when present, indicates that the set occurrence of **set-name** is identified by its **current-of-set-type**. If CURRENT is not specified, the set occurrence of set-name will be selected by the DBCS using the set selection criteria specified in the MEMBER Subentry of set-name related to **record-name**. If the DBCS fails to select the set occurrence, a **set selection failed** condition is returned in DB-STATUS.

Because the set selection mechanism may consume time if it has to traverse several sets in the hierarchy, it should only be used when the programmer wants to locate one record in set occurrence **set-name**. If the programmer wants to perform a search for more than one record, he should invoke a FIND statement without specifying CURRENT, followed by the necessary FIND statements **with** the CURRENT parameter specified.

If the USING clause is not specified, the search is not based on the contents of the field. The record sought (by a search performed in the forward direction) then becomes the first of that type in the set occurrence. The function of such a FIND statement, when the keyword CURRENT is omitted, is to trigger the set selection mechanism and to establish the current-of-set-type. Further FIND statements (with the keyword CURRENT) based on the relative position or contents of the record, will work directly on the set occurrence identified by the current-of-set-type.

Example 1: Occurrence Selected by the Current-of-set-type

Figure 5-8, further on, illustrates the execution of two different forms of the FIND... USING... statement. The **type** diagram of the data structure to which these statements is directed is given below:



The relevant DDL clauses are given below:

```
    RECORD NAME IS A
        LOCATION IS DIRECT DIRLOC1
        ...
    SET NAME IS S ...
    MEMBER IS B ...
        DUPLICATES ARE NOT ALLOWED FOR B-DIVISION
    ...
    SET NAME IS T ...
    MEMBER IS C ...
        SET SELECTION IS
            THRU S OWNER IDENTIFIED BY DATA-BASE-KEY
        THEN THRU T WHERE OWNER IDENTIFIED BY B-DIVISION.
```

The upper portion of Figure 5-8 illustrates the selection of the set occurrence T by the current-of-set-type. The **current-of-set T** is should be C8. To find record C4 which is characterized by the contents "PETER" of its C-NAME field, the programmer writes the sequence:

```
    MOVE "PETER" TO C-NAME.
    FIND C WITHIN T CURRENT USING C-NAME.
```

The DBCS will first retrieve the owner of the set T occurrence identified by the current-of-set-type, then search this set occurrence in the forward direction, comparing the C-NAME field of each record encountered with the C-NAME field in the UWA. The search stops when a match occurs or when the owner is encountered.

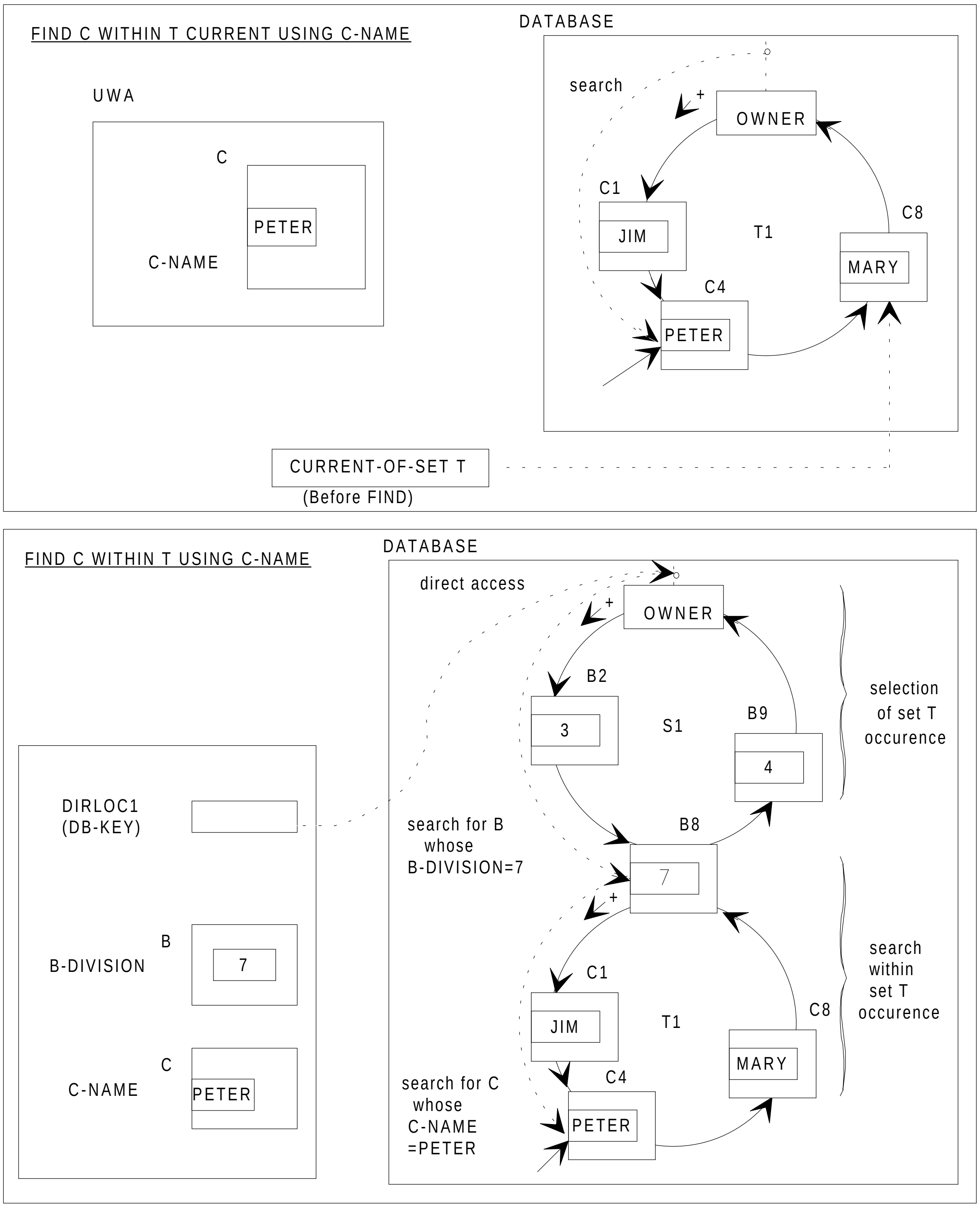

FIND C WITHIN T CURRENT USING C-NAME
DATABASE
UWA
C
PETER
C-NAME
search
+
OWNER
C1
JIM
T1
C8
MARY
C4
PETER
CURRENT-OF-SET T
(Before FIND)
FIND C WITHIN T USING C-NAME
DATABASE
direct access
+
OWNER
B2
3
S1
B9
4
selection of set T occurence
DIRLOC1 (DB-KEY)
search for B whose B-DIVISION=7
B8
7
+
B
B-DIVISION
7
C1
JIM
T1
C8
MARY
search within set T occurence
C
C-NAME
PETER
search for C whose C-NAME =PETER
C4
PETER

Figure 5-8. Sequential Access to a Member Within a Set Occurrence (CONTENTS)

Example 2: Occurrence Selected by the Set Selection Criteria

The lower portion of Figure 5-8 illustrates the selection of a set occurrence using the set selection criteria. The search is for the same record, C4. The sequence of statements becomes:

```
    MOVE SAVE-DBK TO DIRLOC1.  (data-base-key of A4)
    MOVE 7 TO B-DIVISION       (unique identification
                                of B8 within S1)
    MOVE "PETER" TO C-NAME     (identification of C4
                                within T1)
    FIND C WITHIN T USING C-NAME.
```

The DBCS will retrieve A4 by direct access using the data-base-key, then search the set S occurrence whose owner is A4, comparing the B-DIVISION field of each record B encountered with the B-DIVISION field in the UWA. When B8 is found, the set selection is terminated. With B8 being considered as the temporary current-of-set T, the result the processing is then identical to that of the first example.

NOTE: The set T occurrence of the first example could have been identified by the following sequence:

```
    FIND DB-KEY IS SAVE-DBK.
    MOVE 7 TO B-DIVISION.
    FIND B WITHIN S CURRENT USING B-DIVISION.
```

which is equivalent to the implicit set selection of the second example.

5.1.7.2 An Example with Variable Record Length

When the record sought is of variable record length, the length is derived from the UWA contents of the field. In the examples below, the record length is derived from the UWA contents of field R01F01, even though this field is not specified in the USING field sublist.

Consider the following Schema DDL description:

```
                          SCHEMA DDL

  RECORD NAME IS R01
  01  R01F03  TYPE CHARACTER 1.
      ...
  01  R01F01  TYPE SIGNED BINARY 15.
  01  R01F02  TYPE CHARACTER 30 DEPENDING ON R01F01.
```

And the following Subschema DDL:

```
                        SUBSCHEMA DDL

  RECORD NAME IS R01.
```

In the light of the DDL and SDDL above, the following statements are equivalent:

```
  FIND R01 WITHIN ...USING R01F01 R01F02.
  FIND R01 WITHIN ...USING R01F02.
```

In other words, the length of the record sought is derived from the **UWA** contents of field R01F01.

```
FIND R01 WITHIN ...USING R01F03.
```

The field R01F01 must be filled in the UWA record to determine the real length of the variable record. This must be done before executing the FIND verb.

5.1.7.3 Search for Duplicates from the Current-of-set-type

The following statement:

```
    FIND DUPLICATE WITHIN set-name USING identifier ...
```

locates, in the set occurrence **set-name** which is identified by its current-of-set-type, the member record which:

- has the same type as the current-of-set-type
- and whose "identifier" fields have the same contents as those of the same fields in the current-of-set-type.

The search proceeds in the forward direction, starting from the current-of-set-type, which cannot be the owner of the set occurrence. The comparison is made with the fields of the current-of-set-type in the database, not with the fields in the UWA record.

If the owner is encountered before the record sought is retrieved, a **record not found** condition is returned. This does not exclude the presence of duplicates in the portion of the set between the owner and the current-of-set-type.

Figure 5-9 illustrates the action of this statement.

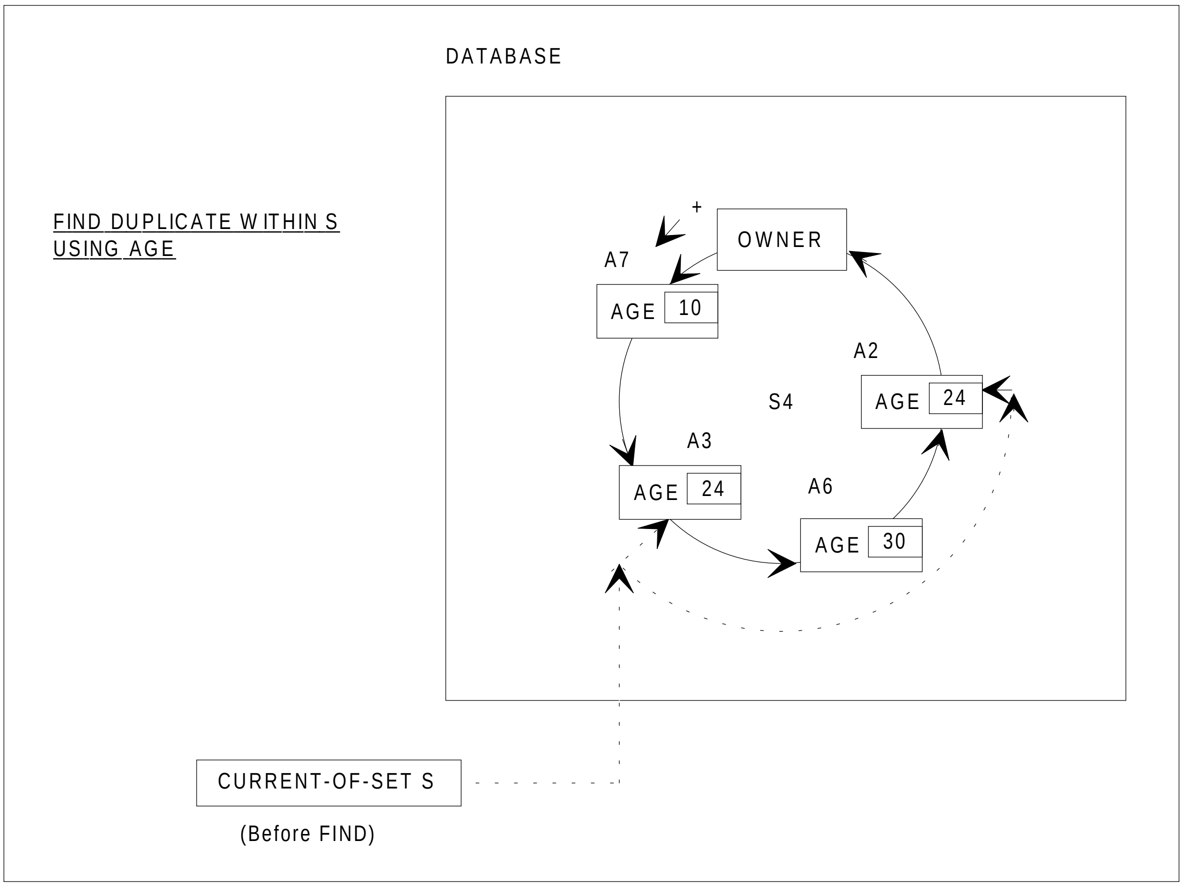


Figure 5-9. Sequential Access to a Member Within a Set Occurence (DUPLICATES)

5.1.7.4 An Example with a Variable Length Record

In the case of a variable length record, the length of the record sought is derived from contents of the current-of-run-unit of field R01F01 **in the database**, even if this field is not specified in the USING field sublist.

The following statements are therefore equivalent:

```
FIND DUPLICATE WITHIN ...USING R01F01 R01F02.
FIND DUPLICATE WITHIN ...USING R01F02.
```

5.1.8 Sequential Access within an Index File using a Relative Position

The record to be retrieved is defined by its key value, regardless of its relative position in the area. This position is defined in the RECORD Entry of Schema description. The position of the record may be relative to either:

- the value of the key (FIRST, LAST, positive or negative ordinal)
- the current record of the key (NEXT, PRIOR)

In the same way as the search of an area, the following can be taken into account during the search :

- any record, whatever its type, in which the relevant key is defined (for a multi-record-type key)
- only records of a specific type, other record types being ignored. In the case of a mono-record-type key, **record name** has no effect.

If the record sought is not found when the end of an index is encountered, an **end-of-key** condition is returned in DB-STATUS and, by default, the name of the key is placed in DB-KEY-NAME. The currencies are left in the state that they were in before the FIND statement was invoked.

5.1.8.1 Using a Position Relative to the Value of the Key

In the index containing the key specified by **key-name**, the following statements:

```
FIND FIRST [record-name] WITHIN key-name
FIND LAST  [record-name] WITHIN key-name
```

locate either:

- the FIRST key (with the lowest value)
- or the LAST key (with the highest value),

relative to all records of the key, or relative to records of the record type specified by **record-name**.

Also, in the index containing the key specified by **key-name**, the following statements :

```
FIND integer    [record-name] WITHIN key-name
FIND identifier [record-name] WITHIN key-name
```

locate the **nth** key, relative to all records, or relative to the record-type indicated by **record-name**. **n** is a positive or negative integer defined in the FIND integer statement (first format) or in a numeric field of the UWA ("identifier" of the second format). If **n** is positive, the search is relative to the record whose key value is the lowest in the referenced key. If **n** is negative, the search is relative to the record whose key value is the highest.

5.1.8.2 Using a Position Relative to the Current-of-key

The following statements:

```
FIND NEXT  [record-name] WITHIN key-name
FIND PRIOR [record-name] WITHIN key-name
```

locate, with respect to the "current-of-key-type" of the key specified by "key-name", the next or the prior record of any type (or of the type indicated by "record-name"), according to the key order.

5.1.9 Sequential Access using a Given Key

The following statement:

```
FIND [record-name] FROM key-name
```

locates the record of the record-type specified by **record-name**, whose key value is equal to or greater than the value of suitable fields in the UWA, according to the key order described below :

- if the key is mono-record-type, suitable fields are the fields of the key in the record.
- if the key is multi-record-type and **record-name** is specified, suitable fields are the fields of the key in the record specified by **record-name**.
- if the key is multi-record-type and **record-name** is not specified (in other words, a search of all records whatever their type), suitable fields are the key db-parameters.
- if **record-name** is specified in the statement, the sought record is the first record of the specified type which is encountered.

If there are no records whose key values are greater than or equal to the key value specified in the UWA, the exception condition "no record found to satisfy the record selection" (02400) is returned.

5.1.10 Access to the Owner within a Set Occurrence

The following statement:

```
FIND OWNER WITHIN set-name
```

locates the owner of the set occurrence **set-name** which is identified by its current-of-set-type (see Figure 5-10).

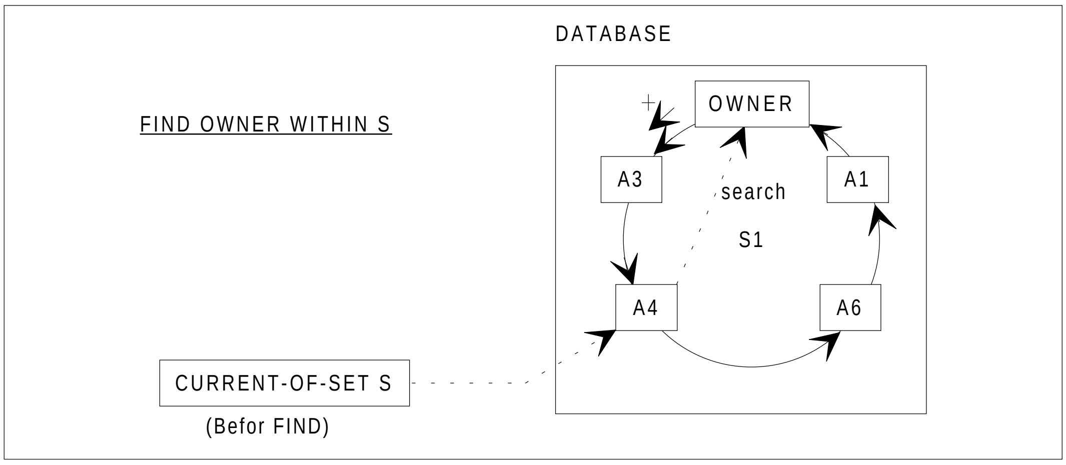
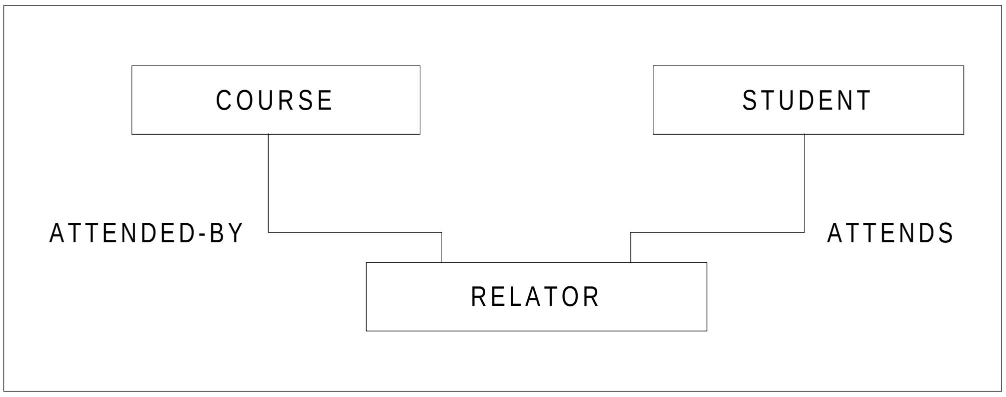


Figure 5-10. Access to the Owner Within a Set Occurrence

5.1.10.1 Example 1: Simple Network

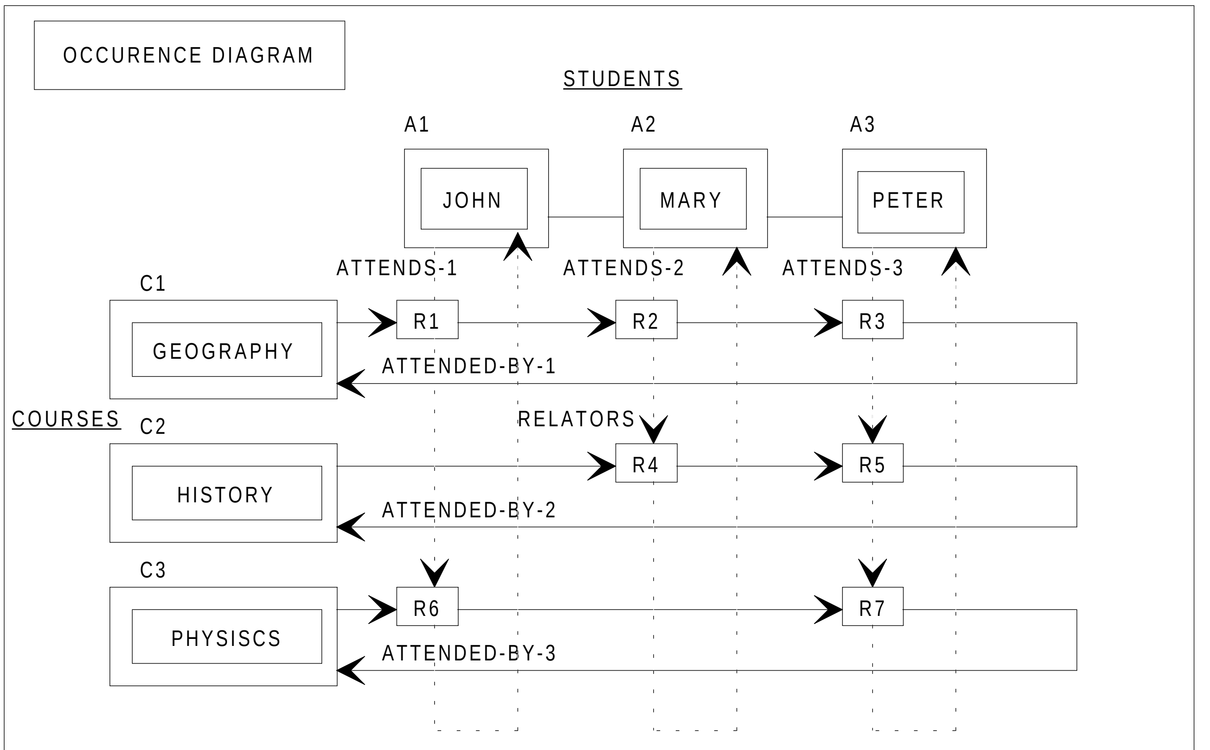
The FIND OWNER statement is particularly useful in a network structure. Let us take as an example the data structure whose type diagram is the following:



In this example, the problem is to retrieve and display the names of all students that attend the HISTORY course. Suppose that COURSE is located in only one area and has a LOCATION of CALC with DUPLICATES NOT ALLOWED, the CALC-key being COURSE-NAME. A typical sequence might be:

```
    MOVE "HISTORY" TO COURSE-NAME.
    FIND ANY COURSE.
           (locate "HISTORY" record)
  LOOP.FIND NEXT RELATOR WITHIN ATTENDED-BY.
           (locate next RELATOR record)
    IF DB-STATUS = "0502100" GO TO END-LOOP.
           (test "end-of-set" condition)
    FIND OWNER WITHIN ATTENDS.
           (locate STUDENT record)
    GET STUDENT.
           (read STUDENT record)
    DISPLAY... GO TO LOOP.
  END-LOOP. ...
```

Figure 5-11, on the following page, illustrates, by an occurrence diagram and a currency chart, the detailed processing of the loop.



"OCCURENCY" CHART

the symbol "-" means that the "currency" is not modified by the statement

DML STATEMENT	CURRENT-OF RUN-UNIT	CURRENT-OF SET ATTENDED BY	CURRENT-OF SET ATTENDEDS
FIND ANY COURSE	C2(HISTORY)	C2(HISTORY)	-
FIND NEXT RELATOR WITHIN ATTENDED-BY	R4	R4	R4
FIND OWNER WITHIN ATTENDS	A2(MARY)	-	A2(MARY)
GET STUDENT	-	-	-
FIND NEXT RELATOR WITHIN ATTENDED-BY	R5	R5	R5
FIND OWNER WITHIN ATTENDS	A3(PETER)	-	A3(PETER)
GET STUDENT	-	-	-
FIND NEXT RELATOR WITHIN ATTENDED-BY (EOS)	-	-	-

Figure 5-11. Access to the Set Owner In a Simple Network

In Figure 5-11, "FIND ANY COURSE" establishes C2 (HISTORY) as the current-of-run-unit and "current-of-set ATTENDED-BY".

"FIND NEXT RELATOR WITHIN ATTENDED-BY" starts from its current-of-set-type, C2, and retrieves R4 which becomes both the current-of-run-unit and the current-of-set of both ATTENDED-BY and ATTENDS.

"FIND OWNER WITHIN ATTENDS" begins its search from its current-of-set-type R4 and retrieves A2 (MARY) which becomes the current-of-run-unit and "current-of-set ATTENDS". The "current-of-set ATTENDED-BY" is not modified and remains R4.

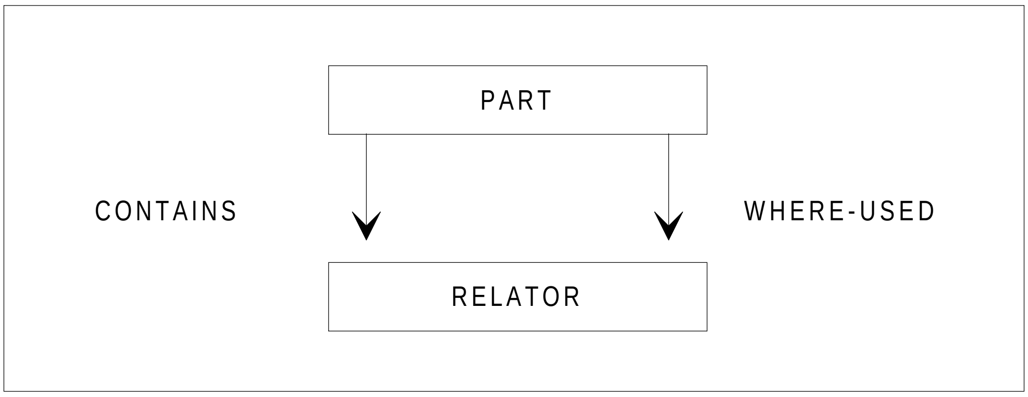
"GET" does not affect currencies.

"FIND NEXT RELATOR WITHIN ATTENDED-BY" starts from its current-of-set-type, R4, and retrieves R5. The processing is repeated until a "FIND NEXT RELATOR WITHIN ATTENDED-BY" returns an **end-of-set** condition which terminates the loop.

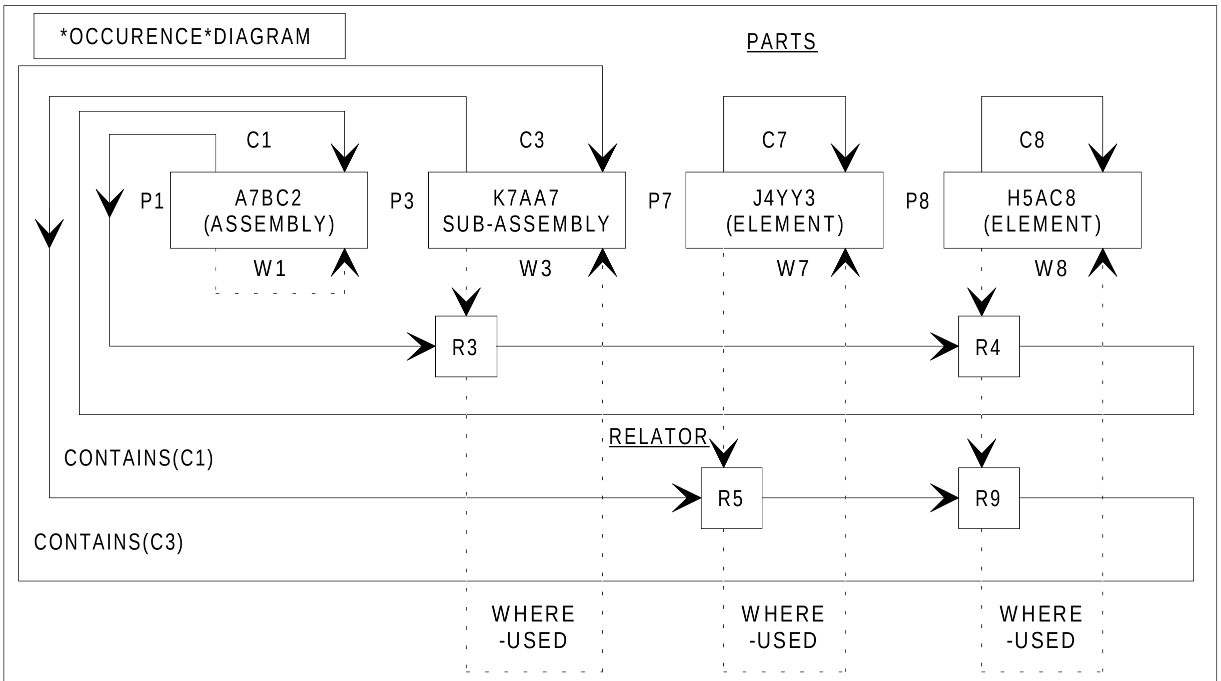
The data structure being symmetrical, the same kind of sequence would enable the programmer to retrieve and display all the courses attended by a given student.

5.1.10.2 Example 2: Complex Network

The problem is more complicated when the two set-types of the preceding data structure have the same owner-type. Let us consider the complex network of a "parts-explosion".



Suppose that PART is located in only one area and has a LOCATION mode CALC with DUPLICATES NOT ALLOWED, the CALC-key being the elementary field PART-NUMBER (see Figure 5-12).



"CURRENCY"CHART

The symbol "-" means that the "currency" is not modified by the statement.

DML STATEMENT	CURRENT-OF RUN-UNIT	CURRENT-OF CONTAINS	CURRENT-OF WHERE-USED
FIND ANY PART	P3	P3	P3
FIND NEXT RELATOR WITHIN CONTAINS	R5	R5	R5
FIND OWNER WITHIN WHERE-USED RETAINING CURRENCY FOR CONTAINS	P7	$\boxed{-}$	P7
GET[PART]	-	-	-
FIND NEXT RELATOR WITHIN CONTAINS	R9	R9	R9
FIND OWNER WITHIN WHERE-USED RETAINING CURRENCY FOR CONTAINS	P8	$\boxed{-}$	P8
GET[PART]	-	-	-
FIND NEXT RELATOR WITHIN CONTAINS(EOS)	-	-	-

Figure 5-12. Access to the Set Owner In a Complex Network

The retrieval sequence needed to find and display the components of part **K7AA7** might be:

```
    MOVE "K7AA7" TO PART-NUMBER.
    FIND ANY PART.
          (locate "K7AA7" part)
  LOOP.FIND NEXT RELATOR WITHIN CONTAINS.
          (locate next RELATOR record)
    IF DB-STATUS = "0502100"
    GO TO END-LOOP.
          (test "end-of-set" condition)
    FIND OWNER WITHIN WHERE-USED
      RETAINING CURRENCY FOR CONTAINS.
          (locate PART, keeping RELATOR
           as "current-of-set CONTAINS")
    GET PART .
    DISPLAY ...
    GO TO LOOP.
  END-LOOP. ...
```

The difference, compared with the case of the simple network, is the presence of the RETAINING Clause in the FIND OWNER statement.

This clause prevents the owner of the WHERE-USED occurrence (for example, P7), from becoming the "current-of-set CONTAINS". Otherwise the latter would identify a new occurrence (C7) of this set and no longer the occurrence (C3) which is searched sequentially. The "FIND NEXT RELATOR WITHIN CONTAINS" would then search C7, starting from P7, (instead of searching C3, starting from R5). The effect of the RETAINING Clause is to maintain R5 as current-of-set of the CONTAINS occurrence, C3.

5.1.11 Retrieval Statements and the Location Mode

Figure 5-13 summarizes which FIND statements may be used according to the record LOCATION mode originally specified.

RETRIEVAL STATEMENT	LOCATION MODE			
	DIRECT	CALC		VIA
		DUP	NO DUP	
FIND DB-KEY IS...	YES	YES		YES
FIND CURRENT...	YES	YES		YES
FIND ANY record-name	NO	YES		NO
FIND DUPLICATE record-name	NO	YES	NO	NO
FIND...USING key-name	YES	YES		YES
FIND...FROM key-name	YES	YES		YES
FIND...WITHIN realm-name	YES	YES		YES
FIND... WITHIN set-name	YES	YES		YES
FIND...WITHIN key-name	YES	YES		YES

Figure 5-13. Retrieval Method Versus LOCATION Mode

5.1.12 Retrieval Statements and Indexes

If the index used in a FIND statement is ATTACHED, the DBCS **will not access the database**.

If the index used is DETACHABLE, the DBCS will access the database in order to check the consistency of the index.

If the data-base-key which is found leads to a missing record in the database, or to a record of a different record type, then the FIND statement is denied and a database exception occurs with the following status code:

```
    DB-STATUS  :  0573615       ( INDEX INCONSISTENCY )
```

If a SET SELECTION Clause activated by the FIND statement invokes a secondary key which is declared in a DETACHABLE INDEX, the DBCS performs a consistency check on the index. This check is a comparison check of the data-base-key of the selected record which is located by the index retrieval path and the database. If the data-base-key which is found leads to a missing record in the database, or to a record of a different record- type, the FIND statement is denied and a database exception occurs with the same status code as shown above.

5.2 GET

5.2.1 General

The GET function causes all or part of a record in the database to be made available in the associated UWA record.

The **object record** of the statement is the current-of-run-unit. Therefore, a GET must therefore be preceded by a FIND. A GET statement does not modify currency indicators.

5.2.2 Global GET

The statement:

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

moves the contents of the current-of-run-unit to the corresponding UWA record (see Figure 5-14). If **record-name** is specified, the DBCS checks that the current-of-run-unit is of this record-type.

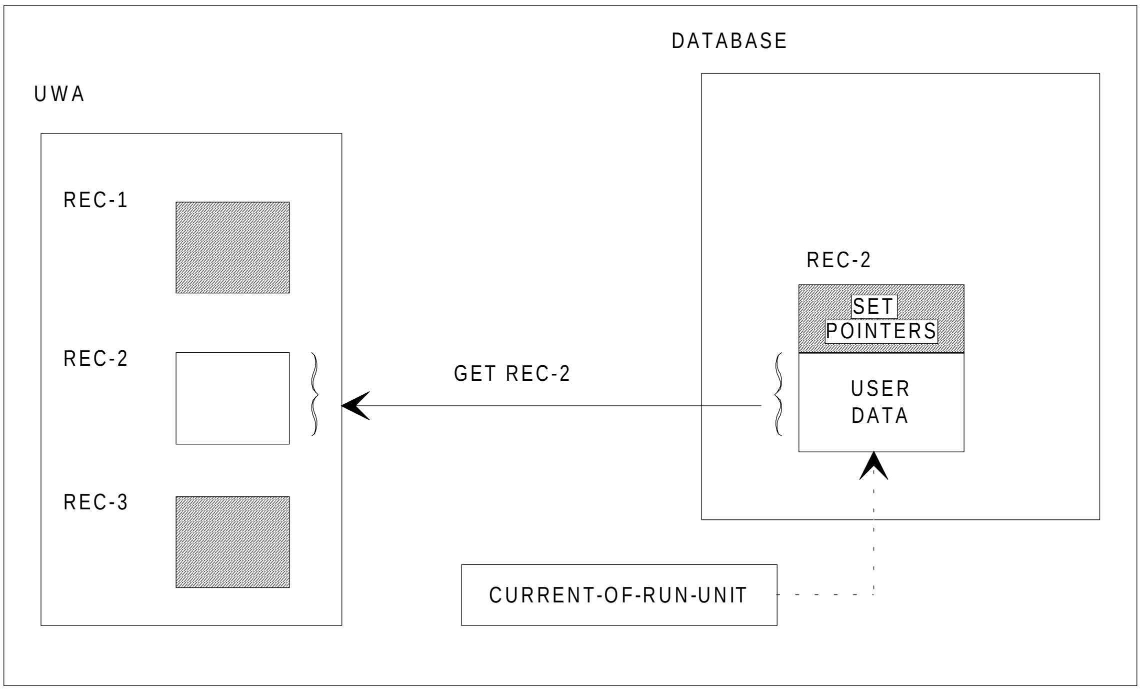


Figure 5-14. Global GET

5.2.3 Selective GET

The statement:

```
GET {identifier} ...
```

causes the contents of the fields specified by "identifier" to be moved from the current-of-run-unit to the associated UWA record (see Figure 5-15). The other fields of the UWA record are not modified.

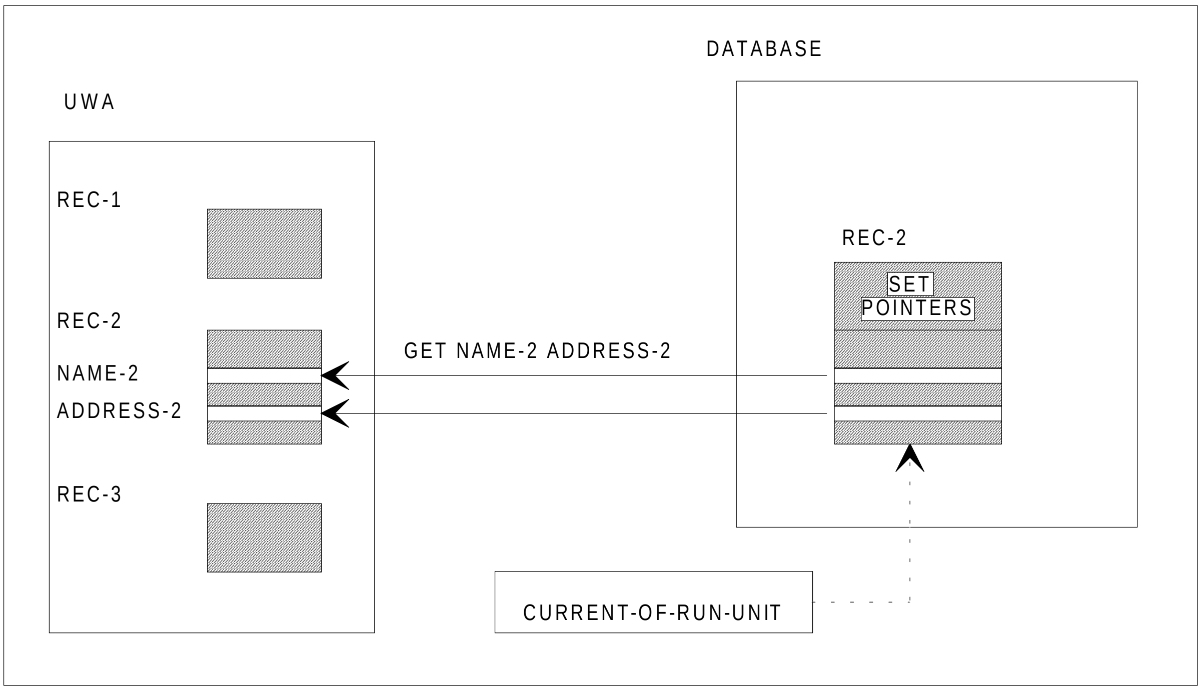


Figure 5-15. Selective GET

5.2.4 An Example with Variable Record Length

Consider the following Schema DDL description:

```
                    SCHEMA DDL

RECORD NAME IS R01
    ...
01  R01F01  TYPE SIGNED BINARY 15.
01  R01F02  TYPE CHARACTER 30 DEPENDING ON R01F01.
```

And the following Subschema DDL:

```
                              SUBSCHEMA DDL

  RECORD NAME IS R01.
```

The following two GET statements yield different results :

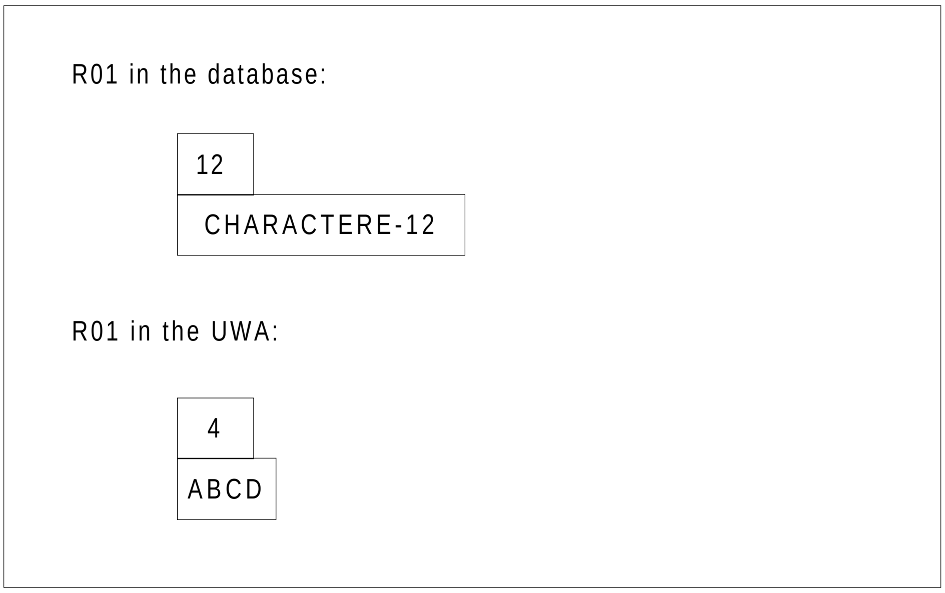
```
  GET R01F01 R01F02.
  GET R01F02.
```

After execution of the first statement, the record length of R01 in the UWA is equal to the length of the current-of-run-unit **in the database**. Following the second statement, the record length of R01 in the UWA is **unchanged** because the control field (R01F01) is not specified in the statement.

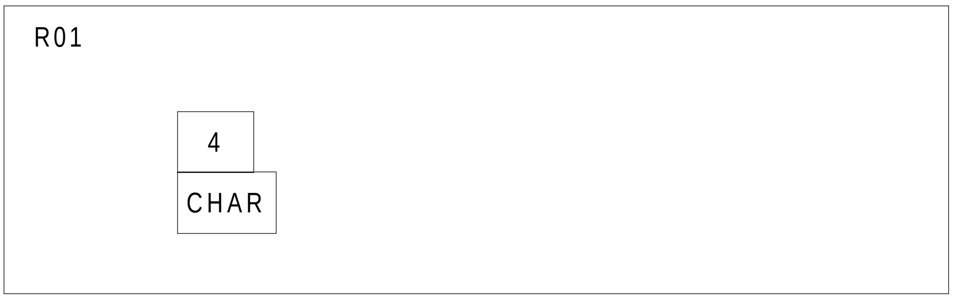
Consider another example:

Suppose we have the following mappings of record R01:

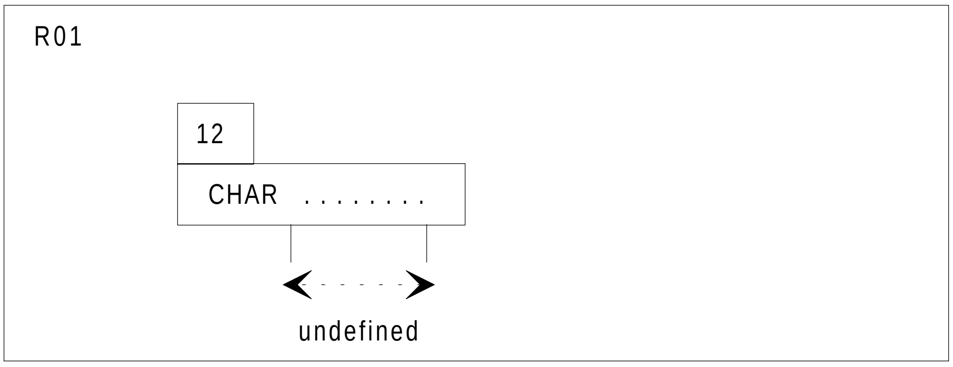
Suppose we have the following mappings of record R01:



Following the execution of "GET R01F02", the mapping of record R01 in the UWA will be:



If "GET R01F01" is then executed,the mapping of record R01 in the UWA will



Therefore, both *control fields* and *depending on fields* should be used at the same time.

5.3 ACCEPT

5.3.1 General

The DML ACCEPT function is an extension to the COBOL ACCEPT verb. The ACCEPT function allows the user to perform the following:

1) read the data-base-keys contained in the currency indicators

2) derive the realm-name which is encoded in a data-base-key

3) read the data-base-key of the NEXT, PRIOR or OWNER of a record within a set

4) read the data-base-key of the NEXT or PRIOR record within a key

5) read DMCL information about an area or a record range from the object schema.

The fifth function permits the parameterization of user routines which process data-base-keys.

The execution of an ACCEPT function does not modify the currency indicators.

The five functions of the ACCEPT statement mentioned above are described in detail in the subsections which follow.

5.3.2 Reading the Data-base-key Contained in a Currency Indicator

The following ACCEPT statements:

```
ACCEPT identifier FROM             CURRENCY.
ACCEPT identifier FROM realm-name  CURRENCY.
ACCEPT identifier FROM record-name CURRENCY.
ACCEPT identifier FROM set-name    CURRENCY.
ACCEPT identifier FROM key-name    CURRENCY.
```

cause the contents of the corresponding currency indicator to be moved to the UWA item referenced by "identifier". This item must be defined with "USAGE IS DB-KEY". If the currency indicator contains a NULL or virtual pointer, a data-base exception occurs.

Figure 5-16 illustrates the action of these statements.

5.3.2.1 Example

This example shows how to save the data-base-key of a record which is being stored. The COBOL sequence is given below:

```
(DATA DIVISION)
   01 SAVE-DBK USAGE IS DB-KEY.
(PROCEDURE DIVISION)
   STORE REC-2.
   ACCEPT SAVE-DBK FROM CURRENCY
```

Figure 5-16 illustrates the action of these statements.

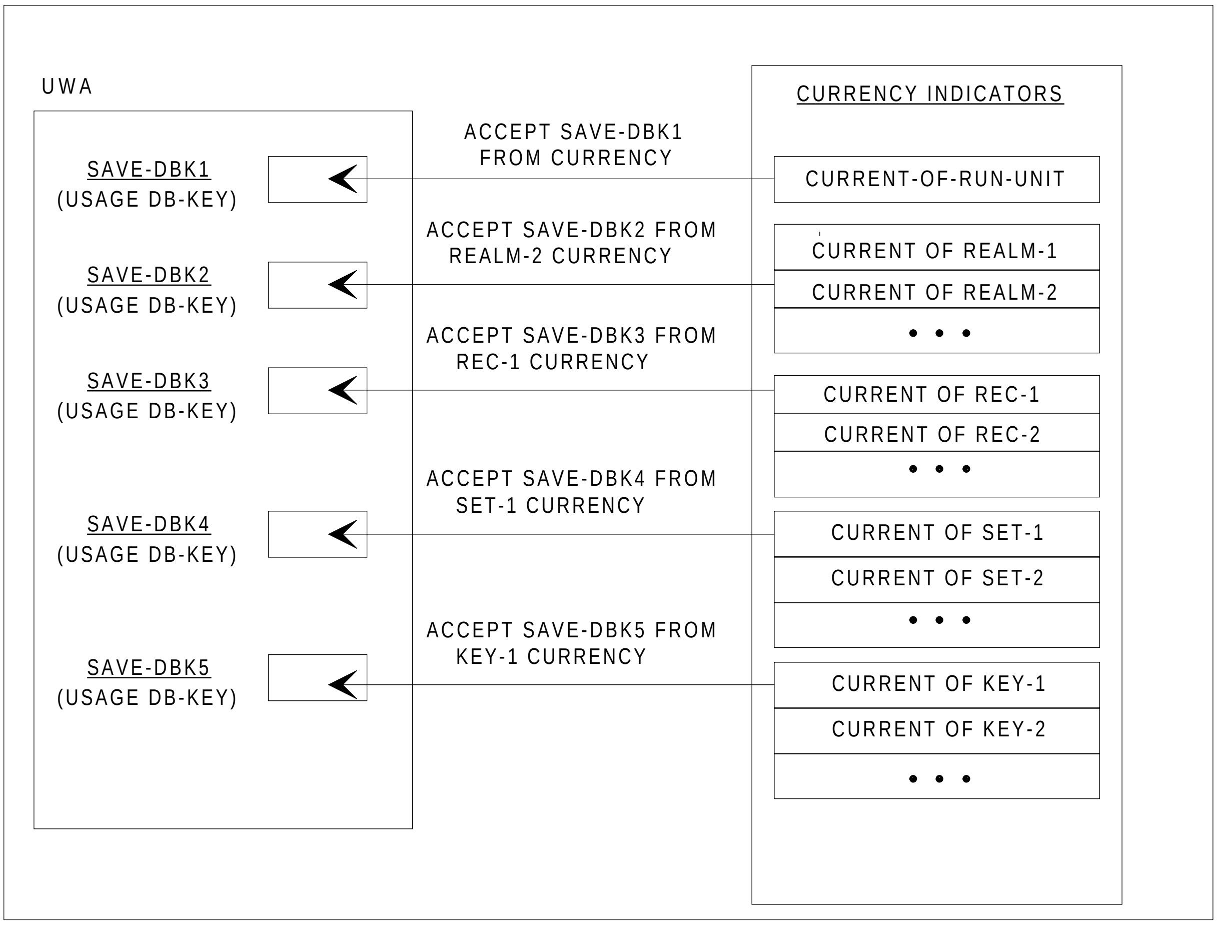


Figure 5-16. Reading of Currency Indicators

5.3.3 Derivation of the Realm-name Encoded in a Data-dase-key

The following ACCEPT statements:

```
ACCEPT identifier-1 FROM               REALM-NAME.
ACCEPT identifier-1 FROM record-name   REALM-NAME.
ACCEPT identifier-1 FROM set-name      REALM-NAME.
ACCEPT identifier-1 FROM key-name      REALM-NAME.
ACCEPT identifier-1 FROM identifier-2 REALM-NAME.
```

extract the area code from a data-base-key and place the corresponding realm-name in the UWA. The data-base-key may be contained in the currency indicators (except for the "current-of-realm" since the realm is known), or it may be supplied in the UWA data item "identifier-2", which is declared with USAGE IS DB-KEY. The definition of the data item "identifier-1" where the realm-name is placed must be large enough to accommodate any REALM-NAME. Its minimum size is 30 (PIC X(30)).

If the currency indicator contains a NULL or virtual pointer, or if the data-base-key supplied in data item "identifier-2" is inconsistent, a data-base-exception occurs.

Figure 5-17, on the following page, illustrates the action of these statements. An example of the use of these statements is given in Example 1 of the paragraph entitled "Reading of DMCL Information", further on in this section.

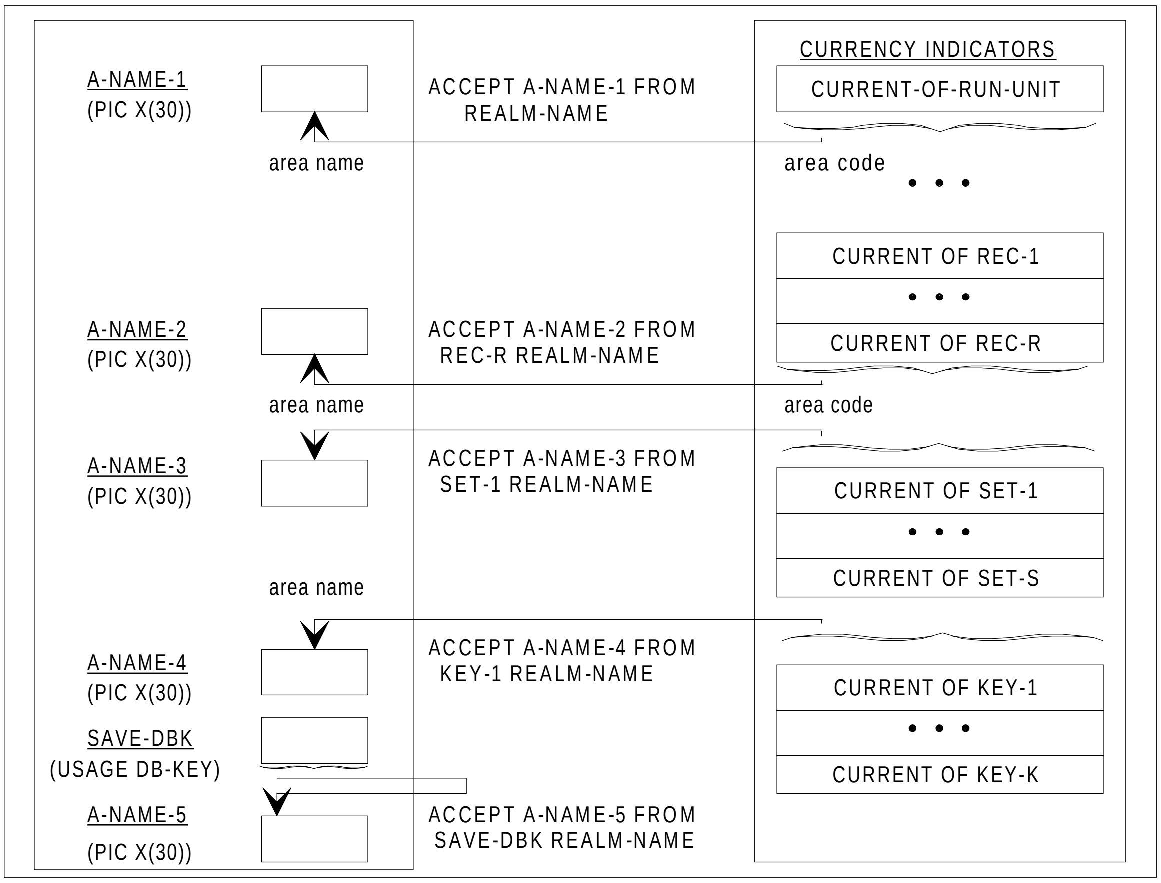


Figure 5-17. Derivation of a Realm-name from a Data-base-key

5.3.4 Reading the Data-base-key of the NEXT, PRIOR or OWNER of a Record within a Set

The following statements:

```
ACCEPT identifier FROM set-name NEXT.
ACCEPT identifier FROM set-name PRIOR.
ACCEPT identifier FROM set-name OWNER.
```

are directed to the record or position identified by the current-of-set of **set-name**. The statements place the data-base-key of the NEXT, PRIOR or OWNER record **within set-name** of this record or position into the UWA data item "identifier". The data item "identifier" must be declared with USAGE DB-KEY.

The current-of-set of **set-name** must not be NULL, otherwise a data-base-exception occurs. It can identify a member (or a position), in which case the NEXT or PRIOR record may be another member or the owner. It can also identify the owner, in which case the NEXT or PRIOR record may be a member, or the owner itself, if the set is empty.

Figure 5-18 illustrates the action of these statements.

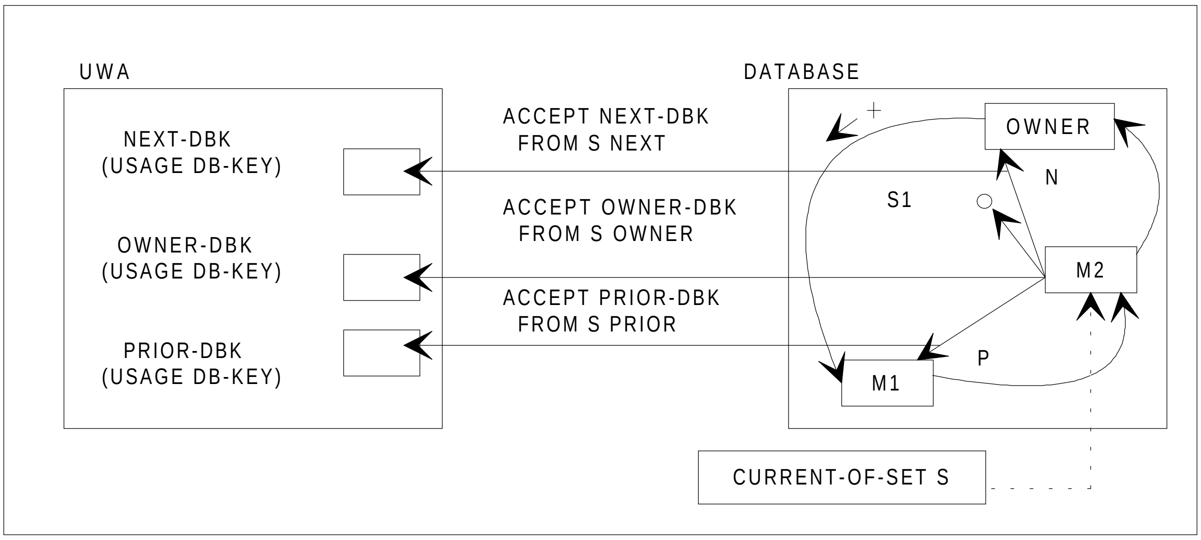
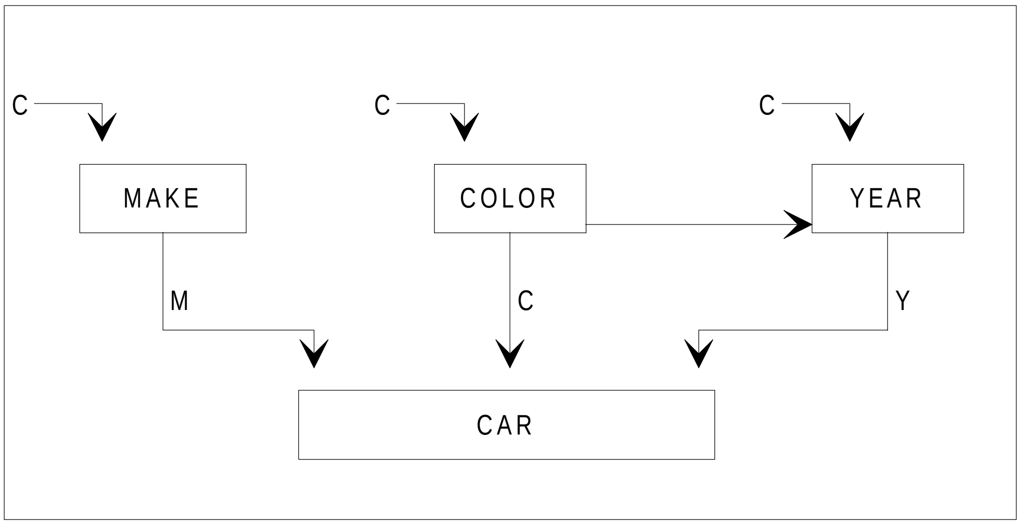


Figure 5-18. Raeding the Data-base-key of the NEXT, PRIOR or OWNER of a Record Within a Set

5.3.4.1 Example: Intersection Of Set Occurrences

Given the data structure below, this example demonstrates how to retrieve all the cars of make ALPHA of the color BLUE and of year 1981.



Assume that the records MAKE, COLOR and YEAR are CALC DUP NOT and are located within one area.

The CAR records sought are those placed at the intersection of three set occurrences:

- the M occurrence M1 whose owner is ALPHA
- the C occurrence C1 whose owner is BLUE
- and the Y occurrence whose owner is 1981

The most symmetrical approach would consist of scanning the area for CAR records (FIND CAR WITHIN area), retrieving for each of them its owner in M, C and Y (FIND OWNER), and checking that these owners are respectively ALPHA, BLUE and 1981.

A first optimization consists of retrieving the CAR records by searching one of the set occurrences (FIND CAR WITHIN set) instead of the area. The set to be privileged is the one having the minimum average number of members per occurrence. We assume that it is set M in this example. The CAR records retrieved will be those of M1.

A second optimization consists of avoiding the retrieval of the owners of C and Y (using FIND OWNER) by comparing the data-base-keys of BLUE and 1981 with the OWNER pointers found in the CAR records (using ACCEPT FROM set OWNER).

The COBOL sequence might be:

```
 (DATA DIVISION)

      01 SAVE-BLUE-DBK USAGE DB-KEY.
      01 SAVE-COLOR-DBK USAGE DB-KEY.
      01 SAVE-1981-DBK USAGE DB-KEY.
      01 SAVE-YEAR-DBK USAGE DB-KEY.

 (PROCEDURE DIVISION)
      MOVE "BLUE" TO COLOR-CALC-KEY.
      FIND ANY COLOR.                        (Locate "BLUE")
      ACCEPT SAVE-BLUE-DBK FROM CURRENCY.    (Save its dbk)
      MOVE "1981" TO YEAR-CALC-KEY.
      FIND ANY YEAR.                         (Locate "1981")
      ACCEPT SAVE-1981-DBK FROM CURRENCY.    (Save its dbk)
      MOVE "ALPHA" TO MAKE-CALC-KEY.
      FIND ANY MAKE.                         (Locate "ALPHA")
LOOP. FIND NEXT CAR WITHIN M.              (Search M1)
      IF DB-STATUS = "0502100" GO END-LOOP.
      ACCEPT SAVE-COLOR-DBK FROM C OWNER.    (save COLOR dbk )

      IF SAVE-COLOR-DBK NOT = SAVE-BLUE-DBK (Compare with
         GO LOOP.                             dbk of "BLUE")
      ACCEPT SAVE-YEAR-DBK FROM Y OWNER.     (save YEAR dbk )
      IF SAVE-YEAR-DBK NOT = SAVE-1981-DBK  (Compare with
         GO LOOP.                             dbk of "1981")
      GET CAR.
      DISPLAY ...
      GO LOOP .
END-LOOP. ...
```

5.3.5 Reading the Data-base-key of the NEXT or PRIOR Record within a Key

The following statements:

```
    ACCEPT identifier FROM key-name NEXT.
    ACCEPT identifier FROM key-name PRIOR.
```

are directed to the record or position identified by the current-of-key of **key-name** and place into the UWA data item "identifier" the data-base-key of the NEXT or PRIOR record within **key-name** of this record or position. "identifier" must be declared with USAGE DB-KEY.

If the current-of-key of **key-name** is NULL, a data-base-exception occurs.

If the current-of-key of **key-name** is the record with the highest key value relative to all records and the first format of the ACCEPT identifier statement shown above is used, the following data-base-exception occurs: end of domain (02100).

If the current-of-key of **key-name** is the record with the lowest key value relative to all records and the second format of the ACCEPT identifier statement shown above is used, the following data-base-exception occurs end of domain (02100).

Figure 5-19 illustrates the action of these statements.

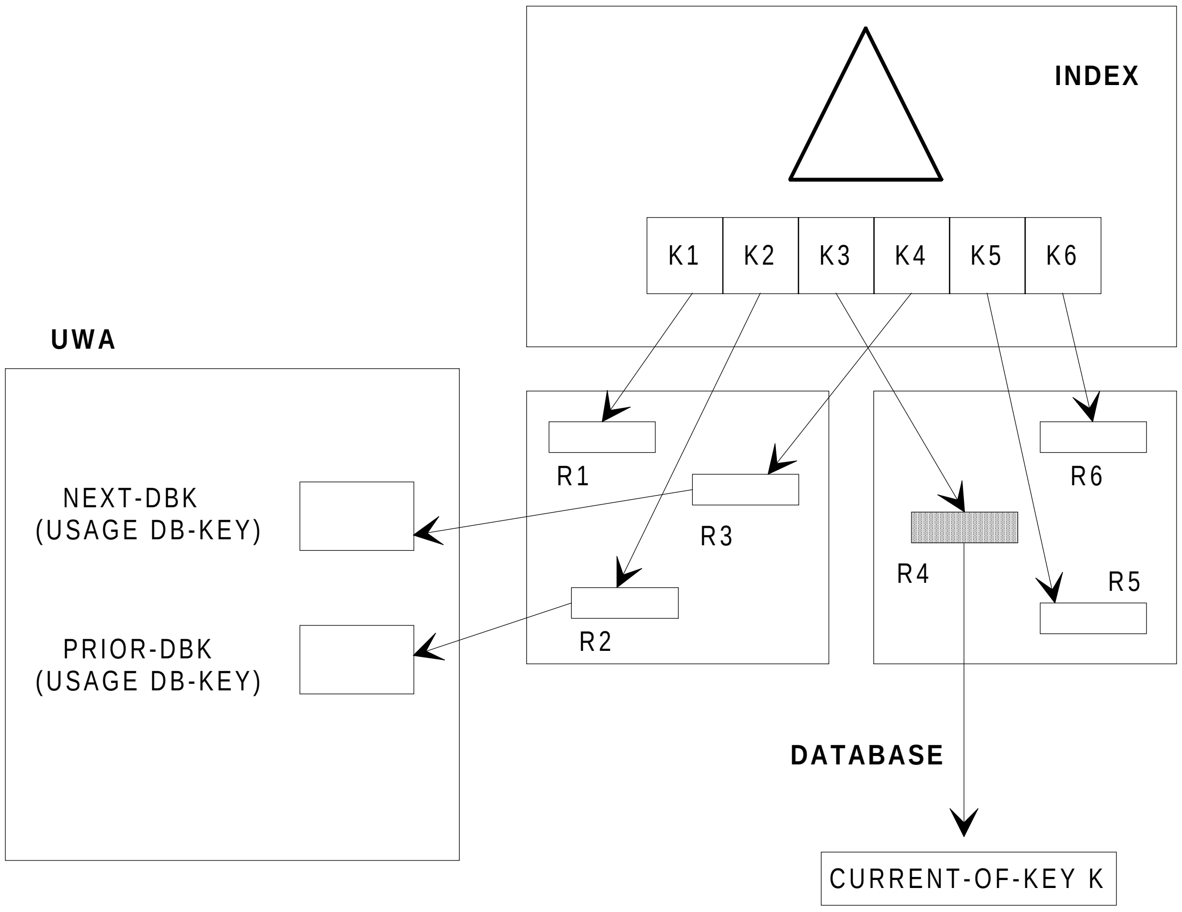


Figure 5-19. Reading the Data-base-key of the NEXT or PRIOR of a Record Within a Secondary Key

5.3.5.1 ACCEPT Statements and Indexes

If the index used during the execution of an ACCEPT FROM key-name statement is **detachable**, the DBCS performs an access to the database in order to check the consistency of the index. If the data-base-key found leads to a missing record in the database, or if the statement leads to a different record type, the ACCEPT statement is denied and a data-base-exception occurs containing the code shown below:

```
DB-STATUS  :  0173615       (INDEX INCONSISTENCY)
```

5.3.6 Reading DMCL Information

The following statement:

```
ACCEPT identifier FROM realm-name LINES-PER-PAGE.
```

reads from the schema the number of lines-per-page of the specified area and places that number in the UWA data item referenced by "identifier". This item must be an integer item with a range at least equal to the range of a comp-1 item.

The following statements:

```
ACCEPT identifier FROM realm-name NUMBER-OF-PAGES.
ACCEPT identifier FROM realm-name NUMBER-OF-PAGES OF record-name.
```

retrieve from the schema the number of pages of the specified area (the first ACCEPT identifier format) or the number of pages of the specified record range within the specified area (the second ACCEPT identifier format), and place that number in the UWA data item referenced by "identifier". This item must be an integer item with a range at least equal to the range of a comp-2 item.

The following statements:

```
ACCEPT identifier FROM realm-name MINIMUM-DB-KEY.
ACCEPT identifier FROM realm-name MINIMUM-DB-KEY OF record-name.
```

retrieve from the schema the minimum data-base-key of the specified area (first ACCEPT identifier format) or the minimum data-base-key of the specified record range within the specified area (second ACCEPT identifier format), and place that data-base-key in the UWA item referenced by "identifier". This item must be defined with USAGE IS DB-KEY.

Figure 5-20, on the following page, illustrates the action of the preceding statements.

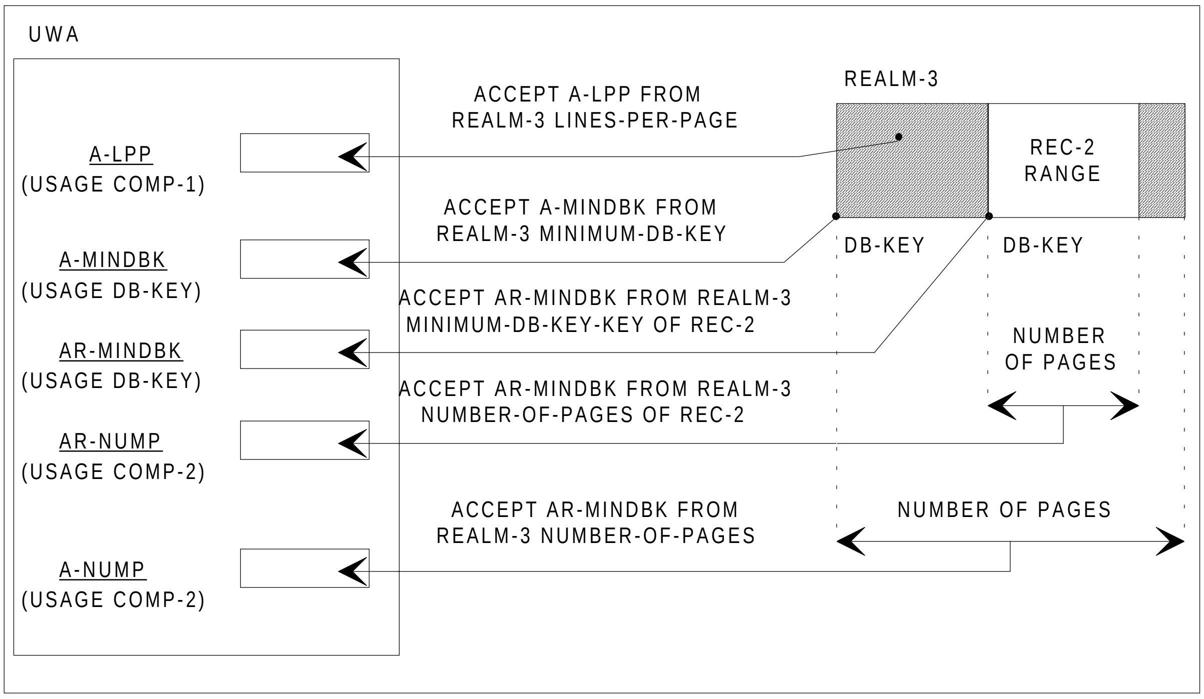

UWA
A-LPP
(USAGE COMP-1)
A-MINDBK
(USAGE DB-KEY)
AR-MINDBK
(USAGE DB-KEY)
AR-NUMP
(USAGE COMP-2)
A-NUMP
(USAGE COMP-2)
ACCEPT A-LPP FROM REALM-3 LINES-PER-PAGE
ACCEPT A-MINDBK FROM REALM-3 MINIMUM-DB-KEY
ACCEPT AR-MINDBK FROM REALM-3 MINIMUM-DB-KEY-KEY OF REC-2
ACCEPT AR-MINDBK FROM REALM-3 NUMBER-OF-PAGES OF REC-2
ACCEPT AR-MINDBK FROM REALM-3 NUMBER-OF-PAGES
REALM-3
REC-2 RANGE
DB-KEY
DB-KEY
NUMBER OF PAGES
NUMBER OF PAGES

Figure 5-20. reading DMCL Information

5.3.6.1 Example 1: Data-base-key Editing

This example shows how to extract the following items from a data-base-key

- its realm-name
- its page number
- its line number (for the purpose of a later edition)

Suppose that there are 2 areas AR-1 and AR-2, and that the data-base-key is in SAVE-DBK.

The COBOL sequence could be:

```
    (DATA DIVISION)
        01 SAVE-DBK USAGE IS DB-KEY.      (data-base-key)
        01 SAVE-AK USAGE IS COMP-2.       (area-key)
        01 DBK-REALM PIC X (30).          (data-base-key
        01 DBK-PAGE-NUMBER USAGE COMP-2.
        01 DBK-LINE-NUMBER USAGE COMP-1.  (components)
        01 A-LPP USAGE IS COMP-1.         (lines-per-page)
        01 A-MINDBK USAGE IS DB-KEY.      (minimum-db-key)
```

```
     (PROCEDURE DIVISION)
             ACCEPT DBK-REALM FROM SAVE-DBK REALM-NAME.
    FIRST-AREA. IF DBK-REALM NOT = "AR-1" GO TO NEXT-AREA.
             ACCEPT A-LPP FROM AR-1 LINES-PER-PAGE.
             ACCEPT A-MINDBK FROM AR-1 MINIMUM-DB-KEY.
             GO TO EXTRACTION.
     NEXT-AREA.    ACCEPT A-LPP FROM AR-2 LINES-PER-PAGE.
             ACCEPT A-MINDBK FROM AR-2 MINIMUM-DB-KEY.
     EXTRACTION.    COMPUTE SAVE-AK = SAVE-DBK - A-MINDBK.
             DIVIDE SAVE-AK BY A-LPP GIVING DBK-PAGE-NUMBER.
                 REMAINDER DBK-LINE-NUMBER.
```

5.3.6.2 Example 2: Placement of DIRECT Records

This example shows how to store a DIRECT record REC-3 on every two pages throughout its range within area AR-4. First, look at Figure 5-21.

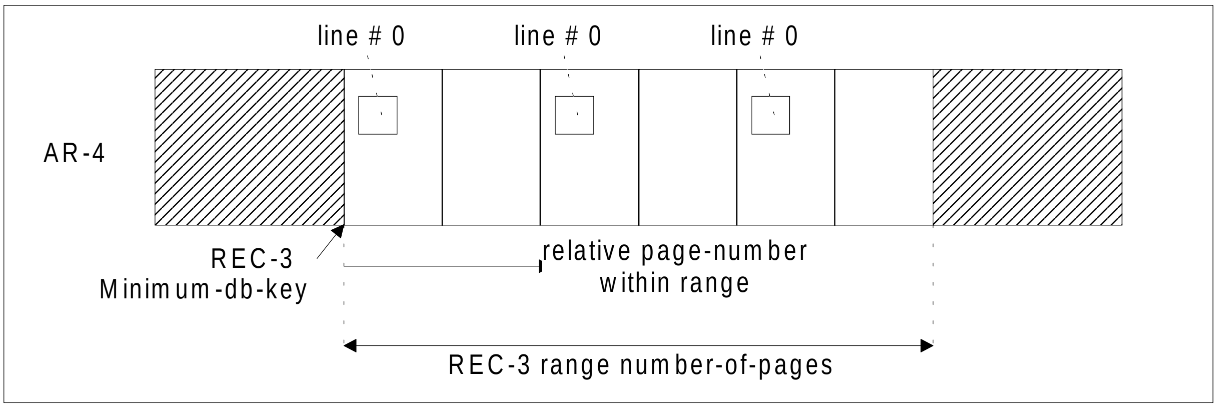


Figure 5-21. Placement Of DIRECT Records

REC-3 is supposed to be located in AR-4 only. Its area-key is supplied in data-base-parameter R3-DIRLOC. REC-3 is, for instance, a functional record containing no data and used only in its capacity of set owner

The COBOL sequence might be:

```
    (DATA DIVISION)
        02 R3-DIRLOC USAGE IS DB-KEY.
           (db-parameter of LOCATION DIRECT generated
            by COBOL compiler)
        01 A4-R3-MINDBK USAGE IS DB-KEY.
          (minimum-db-key of REC-3 within AR-4)
        01 A4-R3-NUMP USAGE IS COMP-2.
          (number-of-pages of REC-3 range within AR-4)
        01 A4-LPP USAGE IS COMP-1.
          (number of lines-per-page of AR-4)
        01 RELATIVE-PAGE-NUMBER USAGE IS COMP-2.
          (relative page number of REC-3 in its range)

    (PROCEDURE DIVISION)
        ACCEPT A4-R3-MINDBK FROM AR-4 MINIMUM-DB-KEY OF REC-3.
        ACCEPT A4-R3-NUMP FROM AR-4 NUMBER-OF-PAGES OF REC-3.
        ACCEPT A4-LPP FROM AR-4 LINES-PER-PAGE.
        MOVE 0 TO RELATIVE-PAGE-NUMBER.

  LOOP.    IF RELATIVE-PAGE-NUMBER NOT < A4-R3-NUMP GO TO END-LOOP.
        COMPUTE R3-DIRLOC = A4-R3-MINDBK +
                    RELATIVE-PAGE-NUMBER * A4-LPP.
        STORE REC-3.
        ADD 2 TO RELATIVE-PAGE-NUMBER.
        GO TO LOOP.

  END-LOOP. ...
```

R3-DIRLOC is loaded with the data-base-key and not with the area-key. The area code part will be ignored by the STORE command. Since there is no choice between areas, AR-4 will be retrieved from the schema.

However if an REC-3 record is used later in a "FIND DB-KEY IS..." statement or in a set-selection-path, the complete data-base-key, as computed in the example, will have to be supplied.

5.4 DATABASE CONDITION (IF)

5.4.1 General

The Database Condition is a simple condition and can be used in COBOL programs with the COBOL IF statement. This statement tests the following conditions:

- **the tenancy condition** (the status of a record with respect to a set occurrence)
- **the membership condition** (the status of a set occurrence)

The evaluation of a Database Condition does not modify currency indicators.

5.4.2 Tenancy Condition

The DB Condition statement (**IF**) :

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

enables the program to determine whether or not the current-of-run-unit is presently one of the following:

- **the owner** of a non-empty occurrence of a specified set-type (or of at least one set-type)
- **an actual member** of a specified set-type (or of at lest one set-type)
- or both of the above

If the current-of-run-unit is neither NULL nor virtual, a data-base-exception occurs.DB

Each parameter of the DB Condition statement are discussed below.

1) Condition with the **set-name OWNER** parameter specified is true when the current-of-run-unit is the owner of a non-empty occurrence of the set-type specified by **set-name**. The record-type of the current-of-run-unit must be declared in the schema as the owner-type of this set-type, otherwise a data-base-exception occurs.

Example: *Test whether a customer has any orders. If so, list the orders.*

IF ORDERS-OF-CUSTOMER OWNER GO TO LIST-ORDERS.

2) DB Condition with **OWNER** parameter specified is true when the current-of-run-unit is the owner of a non-empty occurrence of at least one set-type. The record-type of the current-of-run-unit must appear in the schema as the owner-type of at least one set-type, otherwise a data-base-exception occurs.

3) DB Condition with the **set-name MEMBER** parameter specified is true when the current-of-run-unit is an actual member of an occurrence of the set-type specified by **set-name**. The record-type of the current-of-run-unit must be declared in the schema as a MANUAL OPTIONAL member of this set-type, otherwise a data-base-exception occurs.

Example: *If a record is currently connected to set S, find its owner.*

IF S MEMBER FIND OWNER WITHIN S.

4) DB Condition with the **MEMBER** parameter specified is true when the current-of-run-unit is an actual member of an occurrence of at least one set-type. The set-types considered are those for which the record-type of the current-of-run-unit is declared in the schema as a MANUAL OPTIONAL member. If no such set-type exists, a data-base-exception occurs.

5) DB Condition with the **set-name TENANT** parameter specified is equivalent to either:

   DB Condition set-name OWNER, if the record-type of the current-of-run-unit is declared in the schema as owner-type of this set-type.

   Or *DB Condition set-name MEMBER*, if the record-type of the current-of-run-unit is declared in the schema as a MANUAL OPTIONAL member-type of this set-type.

   If the record-type of the current-of-run-unit is not declared as the owner-type or a MANUAL OPTIONAL member-type of this set-type, a data-base-exception occurs.

6) DB Condition with the **TENANT** parameter specified is the logical "OR" of DB Condition OWNER and DB Condition MEMBER. For this condition to be set-types must exist in the schema either:

   for which the record-type of the current-of-run-unit is the owner type

   and/or for which the record-type of the current-of-run-unit is a MANUAL OPTIONAL member-type.

   Otherwise a data-base-exception occurs.

7) When NOT is specified in the DB Condition, the result of the test is reversed, as shown in the examples below.

Example: *Erase a record if it has currently no members.*

IF NOT OWNER ERASE.

Example: *Connect a MANUAL OPTIONAL record to a set if it is not currently connected*.

IF NOT S MEMBER CONNECT TO S.

NOTE: The Tenancy condition cannot be used to interrogate the "type" data structure described in the schema. It deals only with occurrences of the elements of the data structure. For instance the purpose of "IF OWNER" is not to test if the record-type of the current-of-run-unit appears in the schema as the owner type of any set-types, but rather to test if the current-of-run-unit is currently the owner of at least one non-empty set occurrence. In any case, the schema data structure is supposed to be known to the programmer.

5.4.3 Membership Condition

The DB Condition statement below:

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

enables the program to determine whether or not the occurrence of the specified set-type, which is identified by its current-of-set-type, currently has any members.

The following rules hold true for this statement:

1) The current-of-set-type must neither be NULL nor virtual, otherwise a data-base-exception occurs.

2) DB Condition set-name IS EMPTY is true if the set occurrence has no members.

3) When NOT is specified, the result of the test is reversed.

4) If the owner of the set occurrence is both the current-of-run-unit and the current-of-set-type, the following tenancy condition:

   ```
   IF set-name OWNER
   ```

   is equivalent to the membership condition :

   ```
   IF set-name IS NOT EMPTY
   ```

5.4.4 Syntax Rule

The results of the DB condition test fall into three categories:

1) If no data-base-exception occurs during processing, the condition is true.

2) If no data-base-exception occurs during processing, the condition is false.

3) If a data-base-exception occurs during processing, the condition cannot be tested.

As the DB Condition is only a 2-way branch, categories 2) and 3) will be grouped in the second part of the branch. The programmer must test for data-base-exceptions to differentiate the two following cases:

```
-IF set-name OWNER GO TO CONDITION-TRUE.

-IF DB-STATUS NOT = "0000000" GO TO CONDITION-CANNOT-BE-TESTED.
 CONDITION FALSE. ...
```

The program can be simplified if a USE FOR DB-EXCEPTION section is specified in the DECLARATIVES part of the PROCEDURE DIVISION, and if the decision to abort the step in case of data-base-exception during the evaluation of a DB Condition is taken in this DB-EXCEPTION section (see the description of the USE statement in this manual). The sequence becomes:

```
IF set-name OWNER GO TO CONDITION-TRUE.
CONDITION-FALSE. ...
```

6. Data Manipulation Update Functions

This section describes the five data manipulation functions which update the database. These are briefly described below.

The **STORE function** enters a record into the database. It assigns a data-base-key to the record and connects it to the sets of which it is an AUTOMATIC member.

The **MODIFY function** modifies the contents and/or set membership of a record.

The **ERASE function** removes one or more records from the database, after disconnection from all the sets to which the record(s) are connected.

The **CONNECT function** connects a MANUAL or AUTOMATIC OPTIONAL record to a set occurrence.

The **DISCONNECT function** disconnects an OPTIONAL record from a set occurrence.

Each of these functions are fully explained in the subsections which follow.

6.1 STORE

6.1.1 General

The STORE function **enters a record into the database**.

The format of the STORE statement is given below:

```
    STORE record-name
```

The contents of the record are prepared in the associated UWA record.

The STORE function involves two main operations:

- **the assignment of a data-base-key** to the record and its physical placement in the database
- **the logical connection of the record to all the sets in which it is defined as an AUTOMATIC member**. In sets where it is defined as an owner, it is established as the owner of an empty occurrence.

The record is submitted to the NO DUPLICATES control and to validity checks which are defined in the schema. If any condition is not satisfied, a data-base-exception occurs.

Following the successful completion of a STORE statement, the record stored becomes the current-of-run-unit, and its record-name and area-name are put into DB-RECORD-NAME and DB-REALM-NAME respectively. Unless the corresponding currency is specified in the RETAINING phrase, that record also becomes:

- the current of its record-type
- the current of its realm
- the current-of-set-type of all the sets of which it is an AUTOMATIC member or of which

it is the owner

- and the current-of-key type of all the keys on which it is declared.

6.1.2 Physical placement of a record via direct location Mode (see Figure 6-1)

The Schema DDL clauses relevant to the physical placement of a record are shown below:

```
LOCATION MODE IS DIRECT   data-base-parameter-1

       { ANY AREA       }
WITHIN {                } [AREA-ID is data-base-parameter-2]
       { {area-name}... }
```

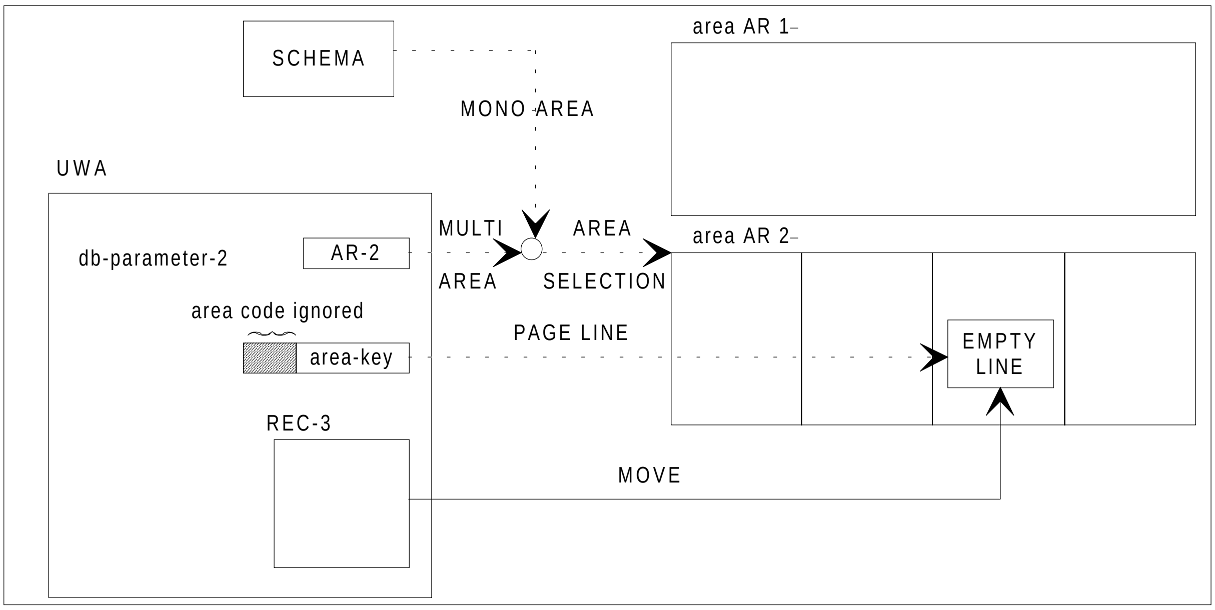


Figure 6-1. Placement of a Record with Direct Location Mode

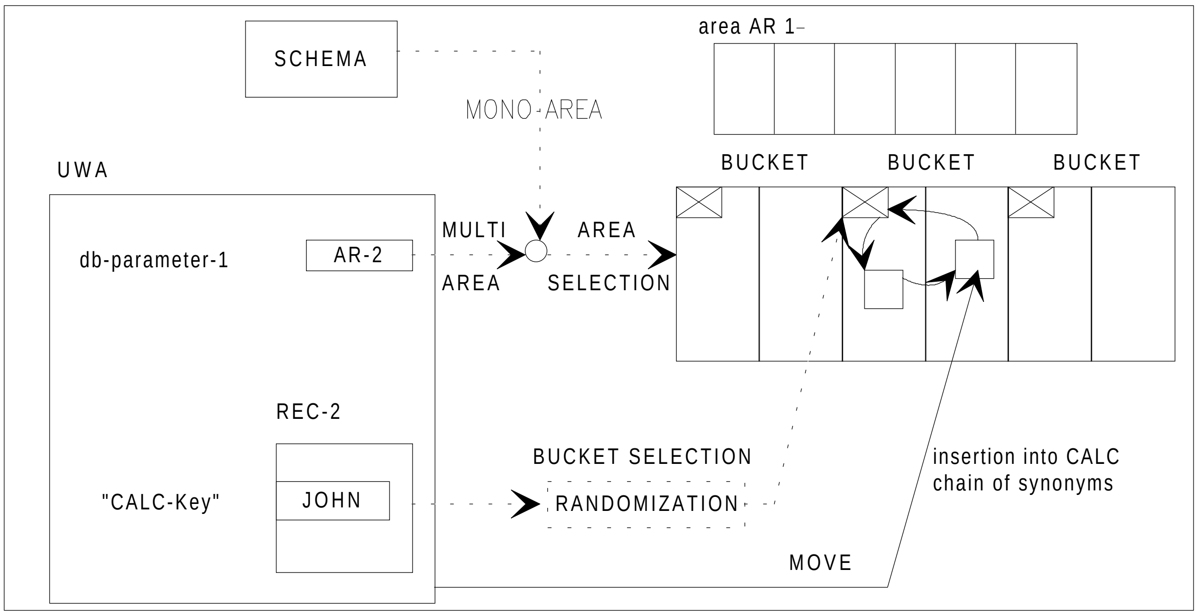


Figure 6-2. Placement of a Record with the CALC method

6.1.2.1 Area Selection

If the record can be located in more than one area, the programmer must initialize db-**parameter-2** in the UWA with the name of the selected area. If there is no choice, the DBCS will retrieve the area from the schema.

6.1.2.2 Area-key Assignment

db-parameter-1 must be initialized with the area-key proposed for the record. The complete data-base-key may be supplied, but the area code portion will be ignored. The area-key determines the page and the line where the record is to be placed. If the line is already occupied or if it is free but there is not enough space left in the page to accommodate the record, the DBCS will search the area from that point in the order of increasing area-keys, until it finds an empty line with enough space in the page. The search stops at the end of the range defined for the record-type and resumes at the beginning of this range up to the starting point. A data-base-exception occurs if the search is unsuccessful.

The result of the processing may be that the area-key eventually assigned to the record is different from the one supplied. After the STORE statement, the ACCEPT statement given below will provide the actual data-base-key:

```
ACCEPT SAVE-DBK FROM CURRENCY.
```

NOTE: This location mode is to be used as infrequently as possible because it does not facilitate the automatic reorganization of the database by a utility routine.

6.1.3 Physical Placement of a CALC Record (see Figure 6-2)

The Schema DDL clauses used to determine the physical placement are given below:

```
LOCATION MODE IS CALC USING {data-base-identifier-1} . . .
                 DUPLICATES ARE [NOT] ALLOWED

       { { ANY AREA         }
WITHIN {                    } [AREA-ID IS data-base-parameter-2]
       { {{area-name} . . . }
```

6.1.3.1 Area Selection

Area selection is performed by the same method used for records whose location mode is DIRECT, as described earlier in this section.

6.1.3.2 Area-Key Assignment

The principle of CALC placement is to divide the area into **buckets** of equal length. A bucket contains one page or a small number of contiguous pages. It is defined by the CALC-INTERVAL Clause in the DMCL. A transformation (called **randomization**) is applied to the contents of one or more elementary fields of the record (consituting its **CALC-key**) in order to determine a bucket of placement in the range of the record-type.

Records which randomize to the same bucket are called **synonyms**. They are chained together by the DBCS, producing as many potential **CALC chains** as buckets. When a bucket contains CALC records of different types, they all participate in the same CALC chain. The DBCS manages any page overflow within a multi-page bucket and any bucket overflow.

The bucket size must be chosen according to the anticipated number of synonyms and the average number of their associated VIA members, if any.

An algorithm is used by the DBCS for randomization, which we will now explain. After concatenation of its elements and "normalization" of sign patterns (DECIMAL items), the CALC-key is processed by the DPS 7 HASH instruction, producing a 32-bit unsigned binary number. This 32-bit number is divided by the number of buckets of the record range, and the result of the division indicates the bucket of placement. In order to obtain an even distribution, it is recommended that the number of buckets be a prime number, or that it should not possess the following small factors: 2, 3, 5, 7, 11, 13, 17, 19.

6.1.4 Physical Placement of Record whose Location Mode is VIA

The Schema DDL clauses used to determine physical placement via record are given below:

```
LOCATION MODE IS VIA set-name

       { {ANY AREA }      }
WITHIN { {area-name} . . .} [AREA-ID IS data-base-parameter-2]
       { AREA OF OWNER    }
```

6.1.4.1 Area Selection

If AREA OF OWNER is specified in the WITHIN Clause, the area where the record is to be placed is determined dynamically when the owner of **set-name** is retrieved. Otherwise the area selection is performed as if the location mode were DIRECT.

6.1.4.2 Area-Key Assignment - General Case

The area-key of the record is determined in relation to the data-base-key of its owner in **set-name**. The set occurrence corresponding to this owner is selected via the set selection criteria. The programmer must therefore initialize, in the UWA, any field necessary for the set selection. If the record is not a MANUAL member of **set-name**, the set selection path will be used again in the second part of the STORE command, when the record is logically connected to this set.

If the record stored is in the same area as its owner and if their record ranges are identical, then it is placed as near its owner as possible. The search of the range for an available area-key starts from the owner and proceeds as in the case of DIRECT location mode. This results in the clustering of the members in the page of the owner or in the pages close to it (see Figure 6-3).

If the record is stored in an area different from that of its owner or if it is stored in the same area but the record ranges are not identical, then a **rule of similarity** is applied (see Figure 6-4). This rule implies that the relative position of the record within its range is proportional to the relative position of its owner within its range. The formula defining this rule is given below:

```
(record  ) (record range)   (owner   ) (owner range)
(area-key)-(min area-key)   (area-key)-(area-key   )
------------ = ------------
 (record range number)       (owner range number )
```

Since this computation produces the same area-key for all possible members of this set occurrence, the DBCS will search the record range from that theoretical point in order to find the next available location. Whatever the result of the calculation, a clustering of member records is achieved.

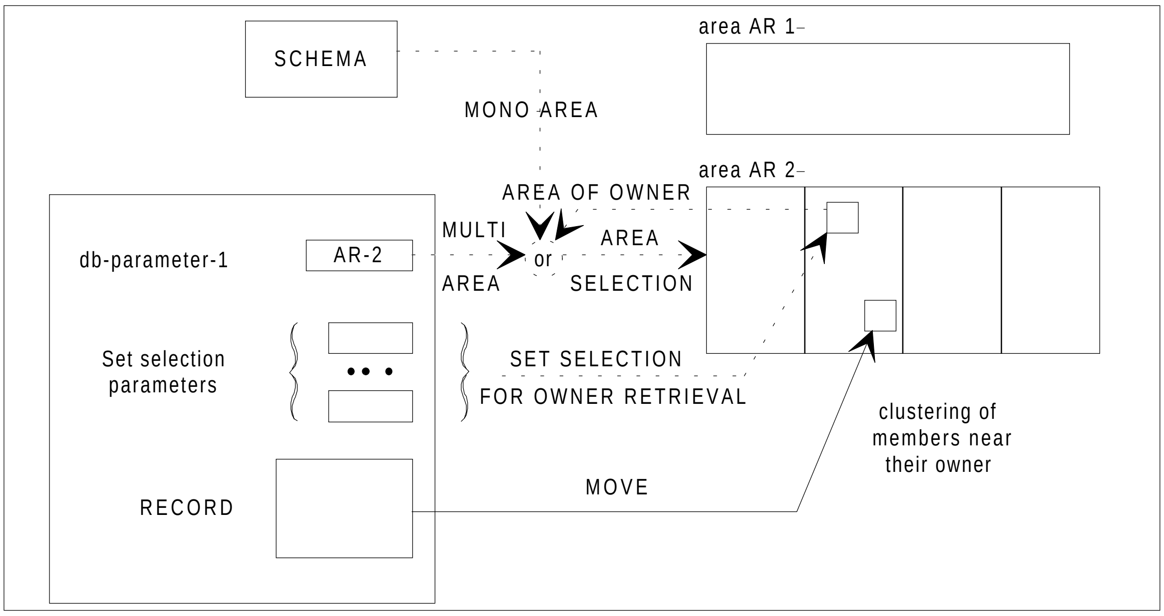


Figure 6-3. Placement of Record with VIA Location Mode (if Owner and Member are in the Same Range)

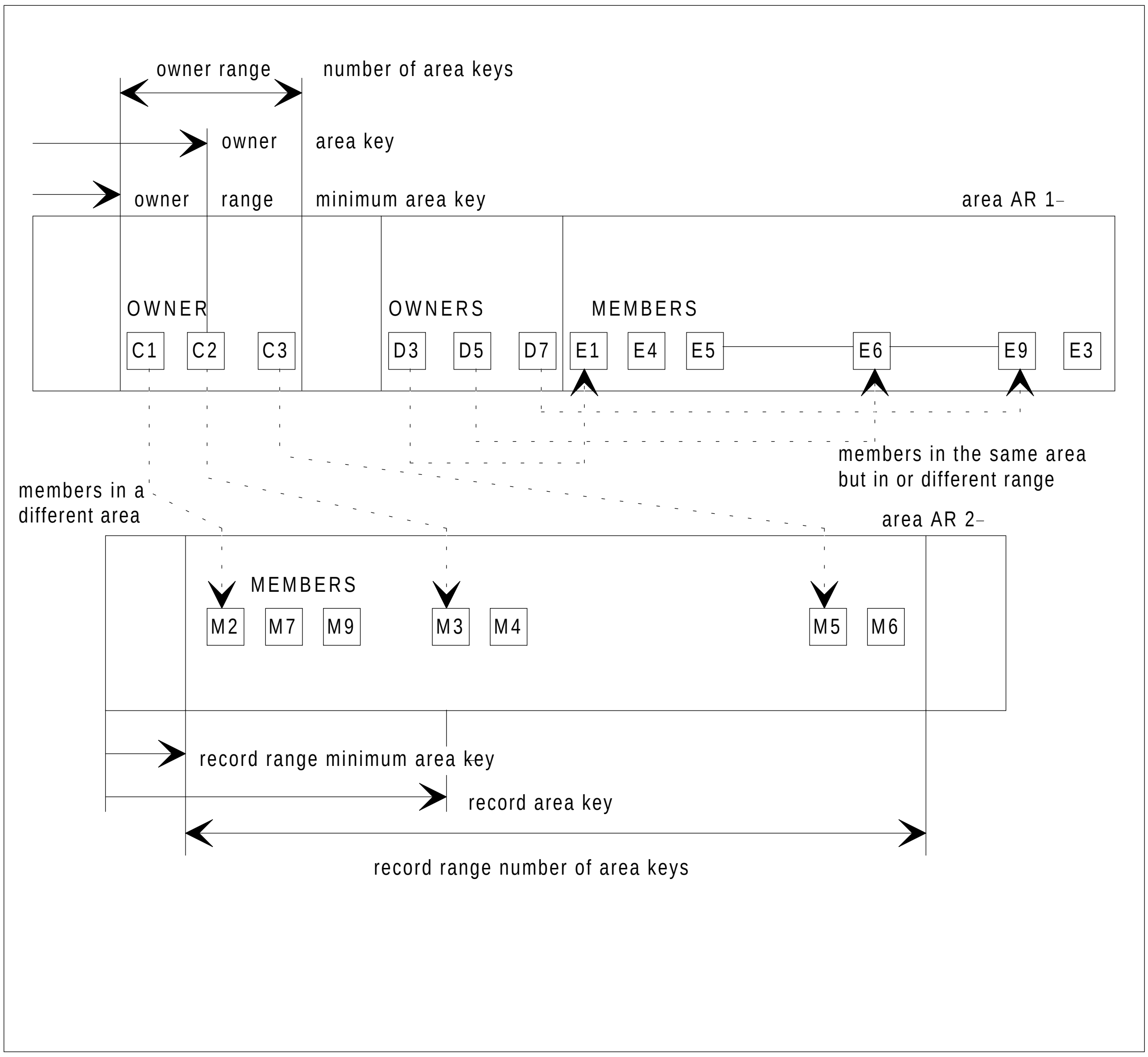


Figure 6-4. Placement of Record with VIA Location Mode (if Owner and Member are in Different Ranges) *Error! Bookmark not defined.*

6.1.4.3 Area-Key Assignment - Special Case

The set to which the VIA phrase is directed is sometimes used to simulate a volatile sequential file. Such a set is likely to spread over many pages.

If the assignment of an area-key were performed as described in the **general case**, above, the search for an empty line from the area-key of the owner (or from that obtained from similarity) would require an access to each page of the cluster every time a member was stored. The DBCS can optimize this process by searching for an empty line from the area-key of the current-of-set-type, provided that it is within the range of the record to be stored (see Figure 6-5). This optimization is only performed when the set to which the VIA phrase is directed has the following DDL characteristics:

- its set selection has only one level identified BY APPLICATION
- its order is NEXT or PRIOR

The optimization requires that the program logic store records in such a way that the current-of-set be stored in the rightmost page of the cluster as illustrated in the lower portion of the figure below.

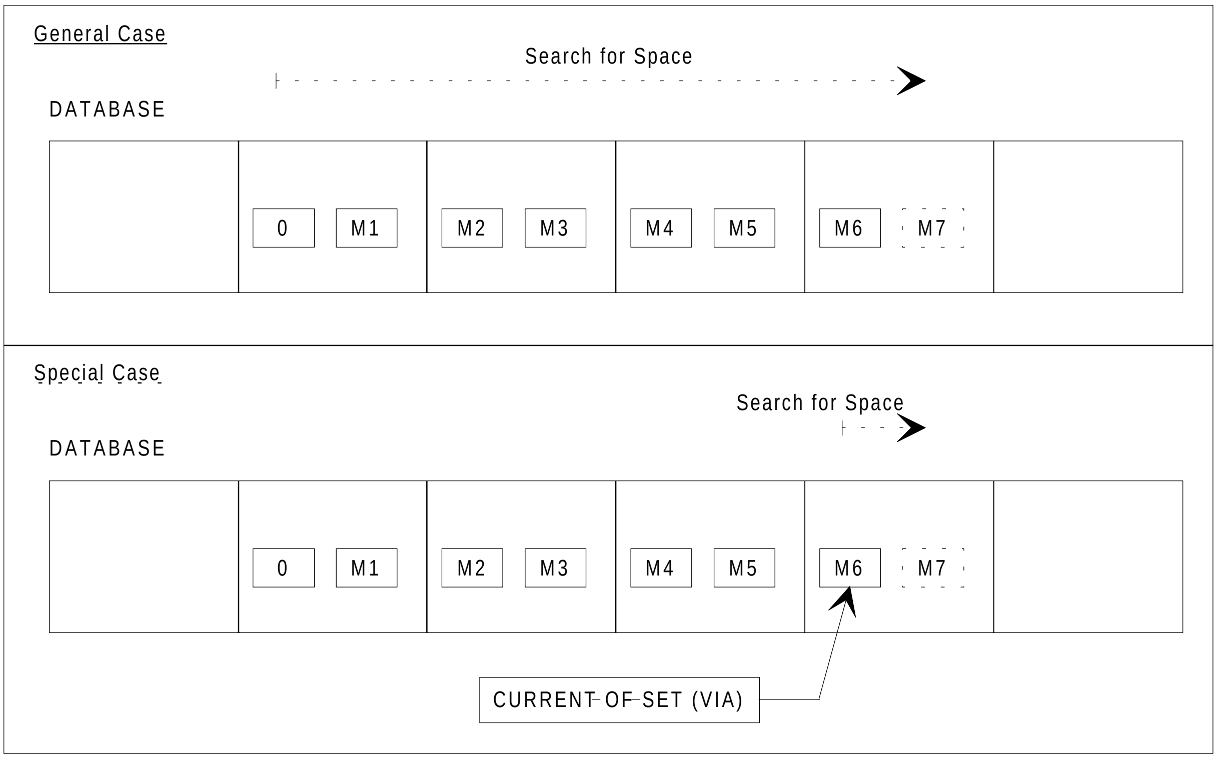


Figure 6-5. Placement of Records using the VIA Location Mode: Searchfor Space

6.1.5 Logical Connection to Sets

A record is connected to all sets in which it is defined as an AUTOMATIC member. The **set occurrence** is selected according to the set selection criteria. **The insertion point** within the set occurrence is determined by the set ordering criteria.

The programmer must therefore initialize the UWA parameters necessary for all the set selection paths prior to the invocation of the STORE statement. Figure 6-6 gives a summary of these parameters.

	DATA-BASE-KEY	**CALA-KEY**	**APPLICATION**
OWNER OF SET OF FIRST LEVEL	db-parameter of LOCATION clause or "EQUAL TO" db-parameter	1) CALC-key of UWA record or "EQUAL TO" db-identifier or db-parameter 2) AREA-ID parameter or "EQUAL TO" db-parameter if there is a choice	"current-of-set-type" (owner or member)
OWNER OF SET OF SECOND AND FOLLOWING LEVELS	identifiers in UWA record or "EQUAL TO" db-identifiers or db-parameters		

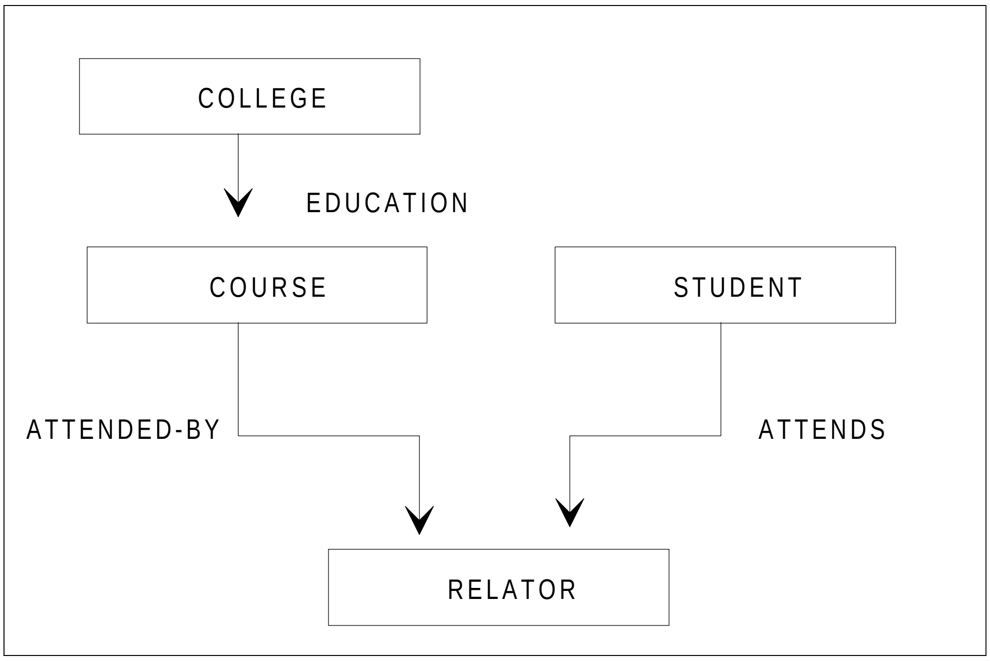
Figure 6-6. UWA Parameters for Set Selection Paths

When the set ordering criteria specify NEXT or PRIOR, a record is inserted next or prior to the current-of-set-type **only if** the set selection path has a single level and the corresponding set occurrence is selected by APPLICATION. In any other case, the insertion point is relative to the owner of the set occurrence selected, which renders NEXT equivalent to FIRST and PRIOR equivalent to LAST.

6.1.6 Example

6.1.6.1 Data Structure

Consider the following type diagram:



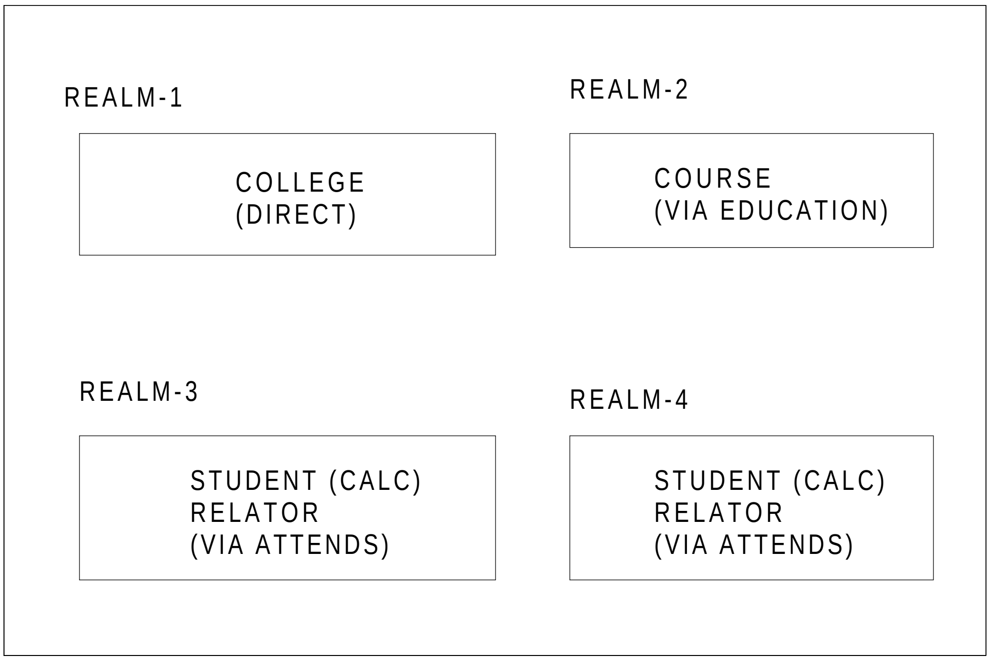
This diagram represents the colleges within a university, the courses given in each college and the students of the university. The RELATOR records represent the courses attended by a given student and the students attending a given course.

The DDL clauses related to the STORE command are shown on the following page.

DDL clauses related to the command STORE

```
RECORD COLLEGE
        LOCATION MODE IS DIRECT DIRLOC
        WITHIN REALM-1.
        ...
    RECORD COURSE
        LOCATION MODE IS VIA EDUCATION
        WITHIN REALM-2.
        ...
        02 C-NAME TYPE IS CHARACTER 12.
         ...
    RECORD STUDENT
        LOCATION MODE IS CALC USING S-NUMBER
            DUPLICATES ARE NOT ALLOWED
        WITHIN REALM-3 REALM-4 AREA-ID IS AR-SELECT.
        ...
        02 S-NUMBER TYPE IS SIGNED UNPACKED DECIMAL 7.
        ...
    RECORD RELATOR
        LOCATION MODE IS VIA ATTENDS
        WITHIN AREA OF OWNER.
        ...
    SET EDUCATION
        OWNER IS COLLEGE
        ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS
            DUPLICATES ARE NOT ALLOWED.
    MEMBER IS COURSE
        INSERTION IS AUTOMATIC RETENTION IS MANDATORY
        KEY IS ASCENDING C-NAME
        SET SELECTION IS
        THRU EDUCATION OWNER IDENTIFIED BY DATA-BASE-KEY.
    SET ATTENDED-BY
        OWNER IS COURSE
        ORDER IS PERMANENT INSERTION IS LAST.
    MEMBER IS RELATOR
        INSERTION IS AUTOMATIC RETENTION IS MANDATORY
        SET SELECTION IS
        THRU EDUCATION OWNER IDENTIFIED BY DATA-BASE-KEY
        THEN THRU ATTENDED-BY WHERE OWNER IDENTIFIED BY C-NAME.
    SET ATTENDS
        OWNER IS STUDENT
        ORDER IS PERMANENT INSERTION IS NEXT.
    MEMBER IS RELATOR
        INSERTION IS AUTOMATIC RETENTION IS MANDATORY
        SET SELECTION IS
        THRU ATTENDS OWNER IDENTIFIED BY APPLICATION.
```

Let us assume that there is no range declaration in the DMCL; the location of the records is graphically shown below:


REALM-1
COLLEGE (DIRECT)
REALM-2
COURSE (VIA EDUCATION)
REALM-3
STUDENT (CALC) RELATOR (VIA ATTENDS)
REALM-4
STUDENT (CALC) RELATOR (VIA ATTENDS)

The subsections which follow provide examples of the storage of various records based on the model above.

6.1.6.2 Storage of a COLLEGE Record (DIRECT, mono-area)

The record is prepared in the UWA. The area-key is computed in COLLEGE-AK by a user-defined algorithm. The storage sequence is:

```
MOVE COLLEGE-AK TO DIRLOC.
STORE COLLEGE.
```

Since there is no choice between areas, the DBCS retrieves "REALM-1" from the schema and assigns to the record the area-key proposed in DIRLOC or the next available one.

The COLLEGE record is then made the owner of an empty EDUCATION occurrence.

6.1.6.3 Storage of a STUDENT Record (CALC, multi-area)

The record, including its CALC-key **S-NUMBER**, is prepared in the UWA. As there is a choice between REALM-3 and REALM-4, AR-SELECT must be initialized. The storage sequences is:

```
MOVE "REALM-3" TO AR-SELECT.
STORE STUDENT.
```

The DBCS consults AR-SELECT in order to determine the area. Then it randomizes the CALC-key S-NUMBER in order to assign an area-key to the record. Before the insertion into the CALC chain, it checks that there is no DUPLICATE STUDENT in that chain.

The STUDENT record is then made the owner of an empty ATTENDS occurrence.

6.1.6.4 Storage of a COURSE Record (VIA, mono-area)

The record is prepared in the UWA. The set selection path of EDUCATION will serve both for the placement of the record and for its connection to the EDUCATION set. It contains only one level, the owner being identified by DATA-BASE-KEY. The complete data-base-key of the relevant COLLEGE record must be supplied, the area-key being insufficient. The storage sequence is:

```
MOVE COLLEGE-DBK TO DIRLOC.
STORE COURSE.
```

The DBCS identifies the owner of the EDUCATION set occurrence by the data-base-key supplied in DIRLOC. The area REALM-2 of the COURSE record is retrieved from the schema, since there is no choice. Since owner and member are not in the same area, the DBCS applies the similarity rule to assign an area-key to the COURSE record.

Being an AUTOMATIC member of the EDUCATION set, the COURSE record is then connected to the EDUCATION set occurrence previously selected. The field C-NAME is used to determine the insertion point in this SORTED set occurrence. Finally, the COURSE record is made the owner of an empty ATTENDED-BY occurrence.

6.1.6.5 Storage of a RELATOR Record (VIA, area of owner)

The record is prepared in the UWA. The set selection path of ATTENDS will serve both for the placement of the RELATOR record and for its connection to ATTENDS. It requires that the relevant STUDENT record be made "current-of-set ATTENDS".

The set selection path of ATTENDED-BY will be useful only to connect the RELATOR record to ATTENDED-BY. This requires that the data-base-key of the relevant COLLEGE record be supplied in DIRLOC and that the field C-NAME of the UWA COURSE record be initialized with the relevant key value.

The storage sequence is, for instance:

```
MOVE "REALM-4" TO AR-SELECT.

MOVE 1227 TO S-NUMBER.         Establish STUDENT record as

FIND ANY STUDENT.              "current-of-set ATTENDS"

MOVE COLLEGE-DBK TO DIRLOC.    Prepare set selection

MOVE "ECONOMY" TO C-NAME.      path of ATTENDED-BY.

STORE RELATOR.
```

The DBCS identifies the owner of the ATTENDS set occurrence by the "current-of-set ATTENDS". The WITHIN Clause of the RELATOR record specifies AREA OF OWNER, so that the record is placed in "REALM-4". Since the STUDENT and RELATOR record ranges are identical (both equal to the area), the RELATOR record is placed near the STUDENT record.

Being an AUTOMATIC member of ATTENDS, the RELATOR record is then connected to the ATTENDS set occurrence previously selected.
As it is also an AUTOMATIC member of ATTENDED-BY, the DBCS moves along the set selection path of ATTENDED-BY. It selects the COLLEGE record by means of the data-base-key supplied in DIRLOC, searches the related EDUCATION set occurrence until it finds a COURSE record in which the contents of field C-NAME are equal to those of the field C-NAME in the UWA. Such a COURSE record identifies the ATTENDED-BY set occurrence to which the RELATOR record is then connected.

6.1.7 Cases when Storage is Impossible

The STORE statement is impossible in the following cases :

- If at least one field of the record is missing in the subschema which has no DEFAULT VALUE.
- If at least one field of the record is missing in the subschema and has a DEFAULT VALUE which is incompatible.

Example:

Consider the following schema description:

```
                                SCHEMA DDL

RECORD NAME   IS  R01
       ...
01     R01F01     SIGNED BINARY 15
                  DEFAULT VALUE IS 100.
01     R01F02     CHAR 20 DEPENDING ON R01F01.
01     R01F03     SIGNED BINARY 31.
```

And the associated subschema:

```
                               SUBSCHEMA DDL

RECORD NAME   IS  R01
01     R01F03.
```

The missing field R01F01 has a DEFAULT VALUE which is incompatible with the maximum value of R01F02 and therefore the following statement is impossible:

```
STORE  R01.
```

and is rejected at compile-time.

- All SET SELECTION Clauses activated by the STORE statement must be acceptable for processing by the DBCS. The criteria for a SET SELECTION clause to be acceptable to the DBCS are listed below :
- the SET SELECTION Clause must be defined (or redefined) BY APPLICATION
- all items named in the SET SELECTION clause must be included in the subschema.

6.1.8 The SET SELECTION Clause with a Detachable Index

If a SET SELECTION Clause activated by the STORE statement invokes a secondary key declared in a DETACHABLE INDEX, the DBCS will perform an index consistency verification which checks the data-base-key of the selected record which is found via the index retrieval path against the database record. If the data-base-key found leads to a missing record in the database, or to a different record-type, then the STORE statement is denied and a data-base-exception occurs with the following status code:

```
DB-STATUS :  1573615       ( INDEX INCONSISTENCY )
```

6.2 MODIFY

6.2.1 General

The MODIFY statement allows the user to:

- modify the contents of some or all fields of a record in the database.
- and/or modify the membership of a record in one or several sets by disconnecting that record from one occurrence of a set and connecting it to another occurrence of this set.

The **object record** of the MODIFY statement is the "current-of-run-unit".

Following the successful execution of a MODIFY statement, provided that the corresponding currency is not specified in the RETAINING phrase, the object record becomes all of the following:

- current of its record-type
- the current of its realm
- the current of all the set-types in which it is a tenant.
- and the current of all the key types on which it is declared.

6.2.2 Modification of the Contents of a Record

The following statement:

```
    MODIFY [record-name]
```

replaces, in the database, the contents of all the elementary fields of the current record of the run-unit with the contents of the corresponding elementary fields of the associated UWA record.

If **record-name** is specified, the DBCS will check that the "current-of-run-unit" is of this record-type.

The following statement:

```
    MODIFY {identifier}...
```

replaces, in the database, the contents of the elementary fields (specified by "identifier") of the current record of the run unit, with the contents of the corresponding elementary fields of the associated UWA record.

From the implicit or explicit list of fields to be modified, the DBCS derives a **sublist** containing the fields whose contents in the database record are actually different from those in the UWA record. (Although it is not recommended, the programmer may simply write "MODIFY" when he intends to modify only one field).

Among these fields, the DBCS distinguishes:

- those fields which are **control-fields**: CALC-key items, sort-key items, no-duplicate-control-fields, fields with validity constraints, secondary key fields.
- those which are **non-control-fields**, as opposed to the control fields mentioned above, which have no special properties relevant to the data structure described in the schema.

The extent to which the database is affected by the MODIFY function depends on what types of control fields are present, if any are present, among the modified fields. If the modified items do not belong to sort-keys or to a CALC-key, only the record is updated. If sort-keys are modified, the DBCS must also reorder the corresponding set occurrences. If the CALC-key is modified, the DBCS must also switch CALC chains and possibly move the record to its new bucket, updating all the set pointers which point to it.

6.2.2.1 No Sort-Key Or CALC-Key Among Modified Fields

If the sub-list contains fields submitted to validity constraints, the DBCS performs the corresponding validity checks. If a condition is not satisfied, the MODIFY statement is denied and a data-base-exception occurs.

If the sub-list contains no-duplicate-control-fields related to one or several sets of which the record is an actual member, the DBCS searches the corresponding set occurrences to check that the new values of the controlled groups of fields do not already exist in any member. If the result of the test is negative, the modification is denied and a data-base-exception occurs.

The contents of the fields to be modified are then moved from the UWA to the database record.

Figure 6-7 illustrates the case of a modification of the whole record. The record to be modified is retrieved by a FIND sequence, the new contents are prepared in the UWA record and then moved to the database by a MODIFY statement:

```
FIND ...
MOVE new-contents TO CUSTOMER
MODIFY CUSTOMER
```

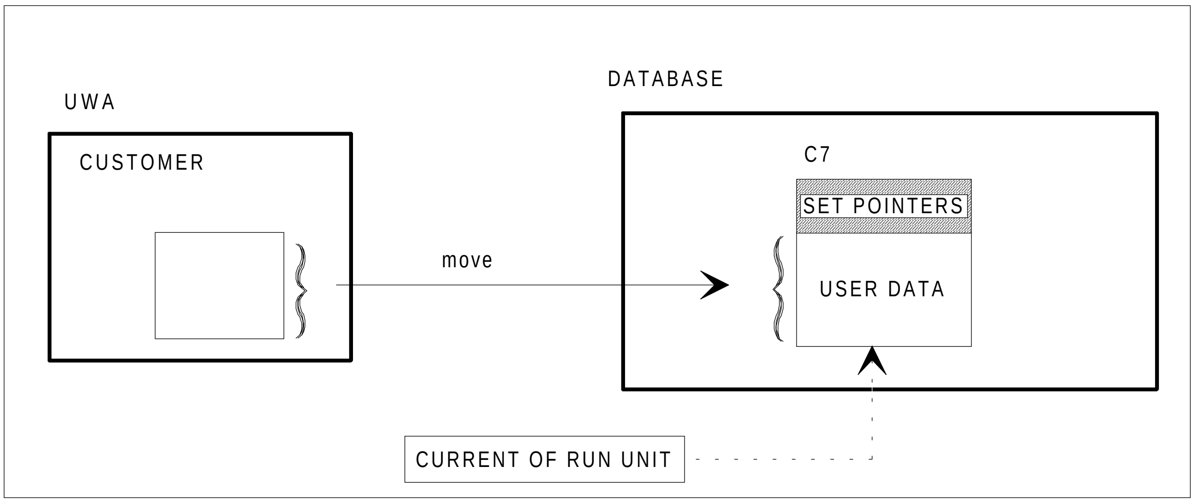


Figure 6-7. Modification of the Whole Record (with no sort-key or CALC-key)

Figure 6-8, below, illustrates the case of a selective modification. The address and phone number of a customer are updated by the following sequence:

```
FIND ...
MOVE new-adress  TO C-ADDRESS
MOVE new-phone-number TO C-PHONE-NUM
MODIFY C-ADRESS C-PHONE-NUM
```

Note that if the retrieval of the record to be modified can be achieved by using FIND statements only, there is no need for a GET statement prior to the MODIFY.

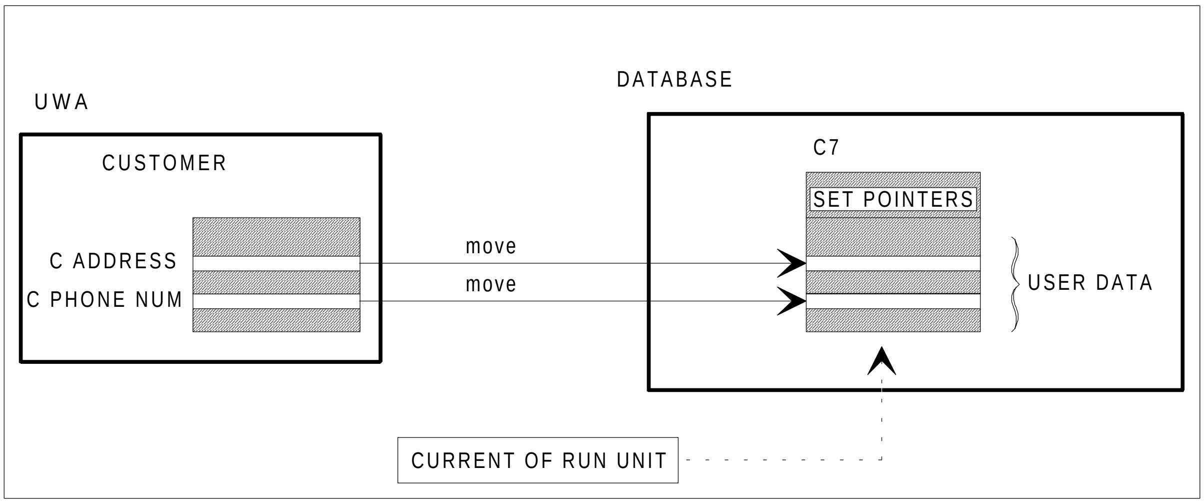


Figure 6-8. Modification of Certain Fields (with no sort-key or CALC-key)

6.2.2.2 Modification Of Sort-keys

If there are sort-key items in the sub-list of modified fields, the DBCS disconnects the record from every set whose sort-key is modified and reinserts it into the same set occurrence according to the set ordering criteria. The SORTED order is thus maintained. If the modification of a sort-key with DUP NOT violates this condition, the modification is denied and a data-base-exception occurs.

Figure 6-9 shows an example of modification of a sort-key. The data structure is composed of the following records CUSTOMER, PART and ORDERS, which are related by sets S and T. The records reside in only one area. CUSTOMER has a LOCATION mode of CALC with DUPLICATES NOT, the CALC-key being C-NUMBER. The insertion mode of ORDERS into S is SORTED with DUPLICATES NOT, the sort-key being ORDER-NUMBER. Its insertion mode into T is FIRST.

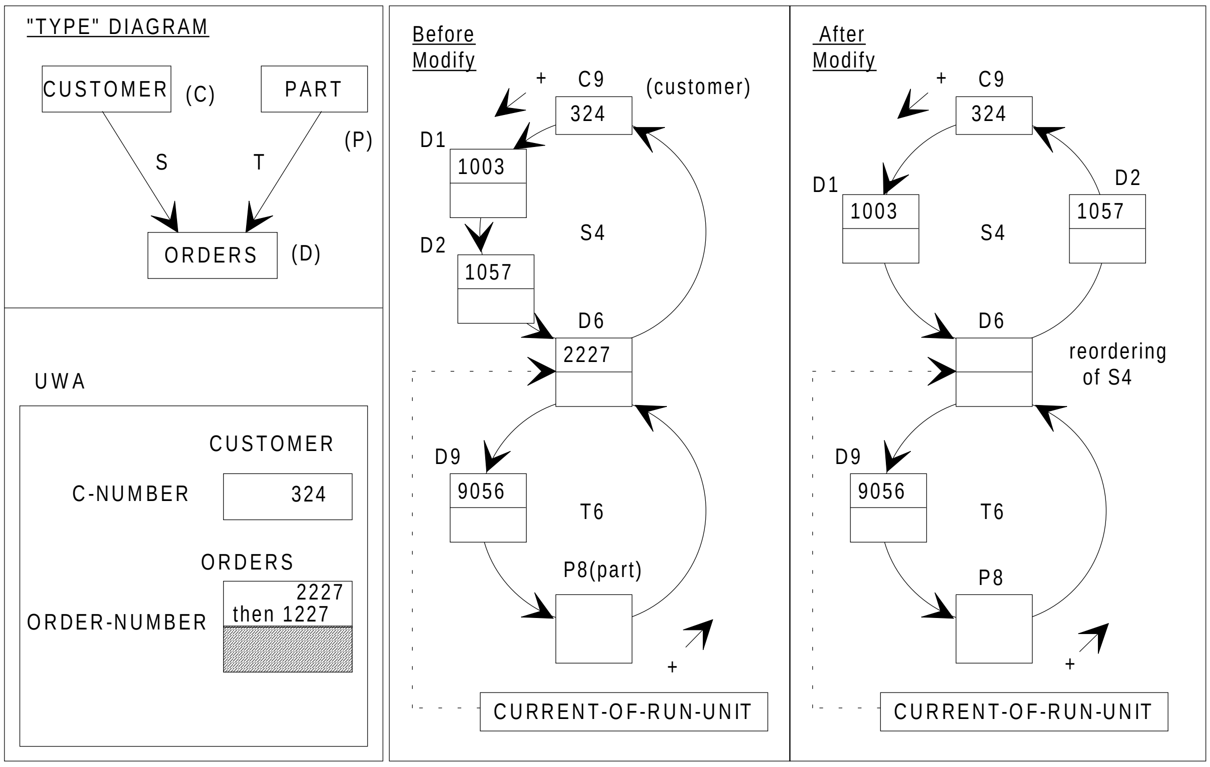


Figure 6-9. Modification of a Sort-key

Suppose that an ORDERS record related to CUSTOMER 324 has been entered into the database with a wrong order number, for example, 2227, instead of 1227, due to a wrongly punched character. The sequence for recovering from this error could be written as follows:

```
MOVE 324 TO C-NUMBER
FIND ANY CUSTOMER
MOVE 2227 TO ORDER-NUMBER
FIND ORDERS WITHIN S CURRENT USING ORDER-NUMBER
MOVE 1227 TO ORDER-NUMBER
MODIFY ORDER-NUMBER
```

In that sequence, the following holds true:

"FIND ANY CUSTOMER" establishes C9 as "current-of-set S".

"FIND ORDERS ..." retrieves D6 within S4.

"MODIFY ORDER-NUMBER" updates the contents of field ORDER-NUMBER in D6, disconnects D6 from set S and reinserts it into the same occurrence S4, between D1 and D2. The position of D6 in set occurrence T6 is not affected.

6.2.2.3 Modification Of Secondary keys

If there are secondary-key items in the sub-list of modified fields, the DBCS removes all the old secondary key values which have been modified from every set, and inserts the new secondary key values into the same indexes. If the modification of a secondary-key value with DUP NOT violates this condition, the modification is denied and a data-base-exception occurs. **This operation is not performed when an index is detachable.**

6.2.2.4 Modification of the CALC-key

If there are CALC-key items in the sub-list of modified items, the DBCS selects the new bucket corresponding to the new value of the CALC-key.

If a DUP NOT condition exists, it is tested. If the test fails, the modification is denied and a data-base-exception occurs.

If the new bucket is the same as the old one, the processing is terminated. If the new bucket is different from the old one, the processing depends on the presence of the MIGRATION IS ALLOWED Clause in the DMCL RECORD Entry (see Figure 6-10, further on in this section).

6.2.2.5 Modification when Migration is Not Allowed

If migration is not allowed, the record is disconnected from the old CALC chain and inserted into the new one. Its data-base-key is not changed.

If, at the necessary time, sufficient room is not available in the page of the record for the DBCS to move into that page the OWL record related to the new CALC chain, the modification is denied and a data-base-exception occurs.

6.2.2.6 Modification when Migration is Allowed

If migration is allowed, the record is disconnected from the old CALC chain, moved to the new bucket (using the same physical placement algorithm as for the STORE statement), and connected to the new CALC chain.

Because, following this modify operation, the data-base-key of the record has changed, the following changes occur:

- all the set pointers which point to the record are updated
- if the data-base-key of the record is a sort-key item in a set, this set is reordered
- the currency indicators pointing to the record are updated. This is independent of the presence of the RETAINING Clause.

Note that only the modified record migrates and that it cannot migrate outside its area.

If the new data-base-key is called for by the program, an ACCEPT FROM CURRENCY statement must follow the MODIFY statement.

6.2.3 Modification of Record Membership in One or More Sets

The following statement:

```
    MODIFY [record-name] ONLY ALL MEMBERSHIP
```

modifies the membership of the current-of-run-unit in all the sets in which the record is an AUTOMATIC MANDATORY member or a MANUAL "NOT FIXED" member currently connected. (The current record must be declared in the schema as member of at least one set.)

The following statement:

```
    MODIFY [record-name] ONLY {set-name} ... MEMBERSHIP
```

modifies the membership of the current-of-run-unit in the sets specified by **set-name**. In each set of the list, the current-of-run-unit must be either an AUTOMATIC MANDATORY member or a MANUAL "NOT FIXED" member currently connected.

When **record-name** is specified, the DBCS checks that the current-of-run-unit is of this record-type.

For every set on the implicit or explicit list, the DBCS proceeds according to steps 1) through 3) below:

1) It disconnects the current record of the run-unit from the set occurrence.

2) It selects a new occurrence of this set according to the set selection criteria. The parameters of the set selection path must have been prepared in the UWA.

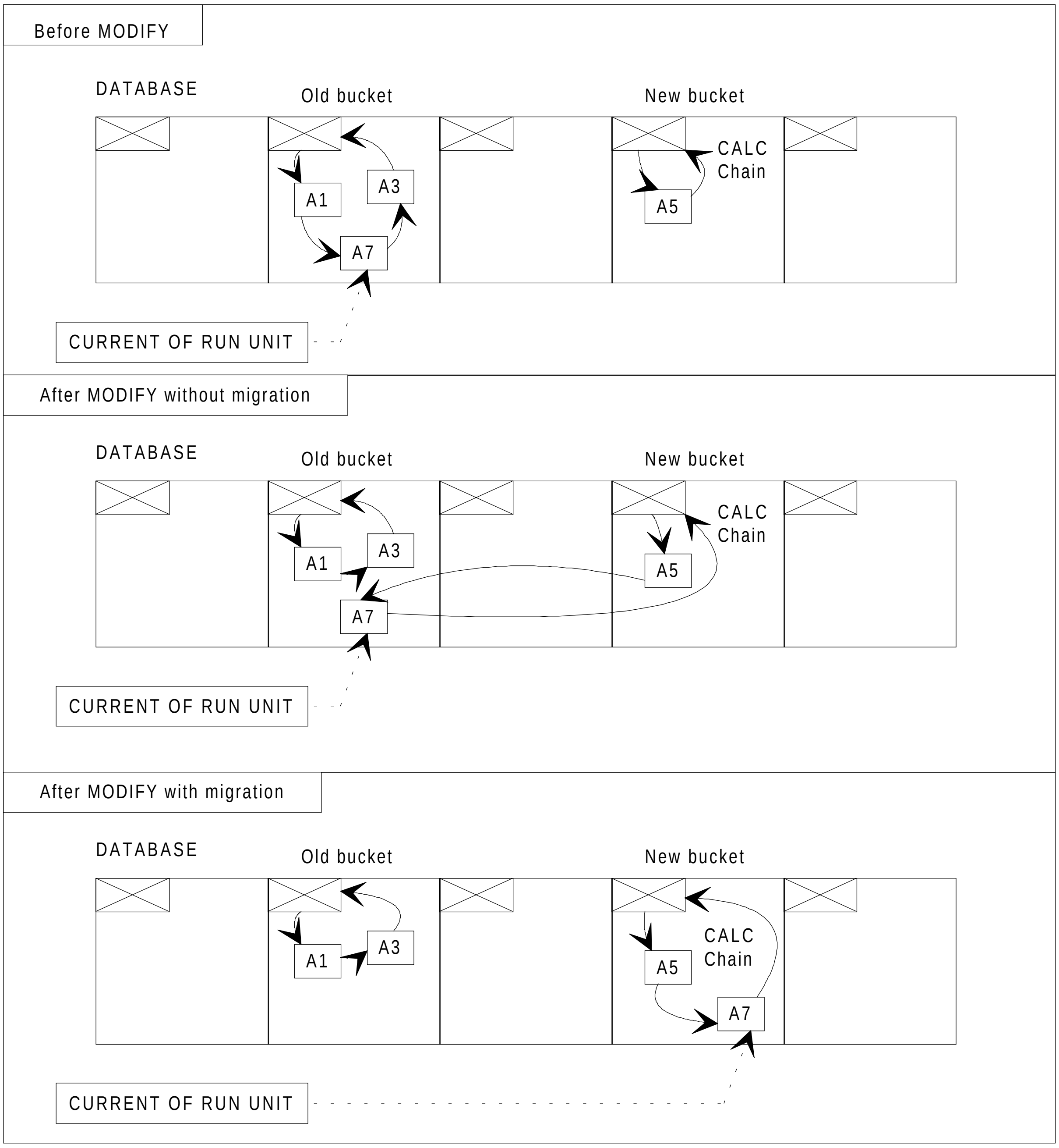


Figure 6-10. Modification of a CALC-key

If the set is the one specified in the VIA phrase of the LOCATION Clause for a record, and if that record is located WITHIN AREA OF OWNER, the DBCS checks that the owner of the selected set occurrence is in the same area as the current record of the run unit. If not, a data-base-exception occurs. Note that only the AREA OF OWNER condition is enforced. Since no record is physically moved, the placement conditions (proximity of owner or "similarity") are no longer satisfied.

3) It inserts the current record of the run-unit into the selected set occurrence according to the "set ordering criteria". If the set is SORTED, the fields involved in the comparison are those of the current record in the database. The associated UWA record is ignored. If the insertion process detects that a NO DUPLICATES condition is not met in the new set occurrence, the modification is denied and a data-base-exception occurs.

6.2.3.1 Example 1: A Record which is Member of One Set and Owner of None

Consider the data structure of Figure 6-11, on the following page. The records are assumed to be located in only one area. The DDL clauses which are relevant to this discussion are given below:

```
RECORD NAME IS DEPARTMENT
    LOCATION MODE IS CALC USING D-NAME
        DUPLICATES ARE NOT ALLOWED
    ...
    02 D-NAME TYPE IS CHARACTER 20.
    ...
RECORD NAME IS EMPLOYEE
    ...
02 E-NUMBER TYPE IS SIGNED PACKED DECIMAL 5.
    ...
SET NAME IS PERSONNEL
    ...
    ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS
        DUPLICATES ARE NOT ALLOWED.
MEMBER IS EMPLOYEE
    ...
    KEY IS ASCENDING E-NUMBER
    SET SELECTION IS THRU PERSONNEL
    OWNER IDENTIFIED BY CALC-KEY.
```

Suppose that employee 325 leaves the MANUFACTURING department and takes on a new job in the SALES department, as a result of a change of orientation in his career. The sequence that reflects this new assignment could be:

```
MOVE "MANUFACTURING" TO D-NAME    Prepare selection path
MOVE 325 TO E-NUMBER              for FIND
FIND EMPLOYEE WITHIN PERSONNEL USING E-NUMBER
MOVE "SALES" TO D-NAME            Prepare selection path
MODIFY EMPLOYEE ONLY PERSONNEL MEMBERSHIP     for MODIFY
```

In this example, the MODIFY statement is equivalent, as far as the set membership is concerned, to an ERASE followed by a STORE. There is, however, a difference, in that a STORE would permit the assignment of a new data-base-key to the record whereas the MODIFY does not change the data-base-key.

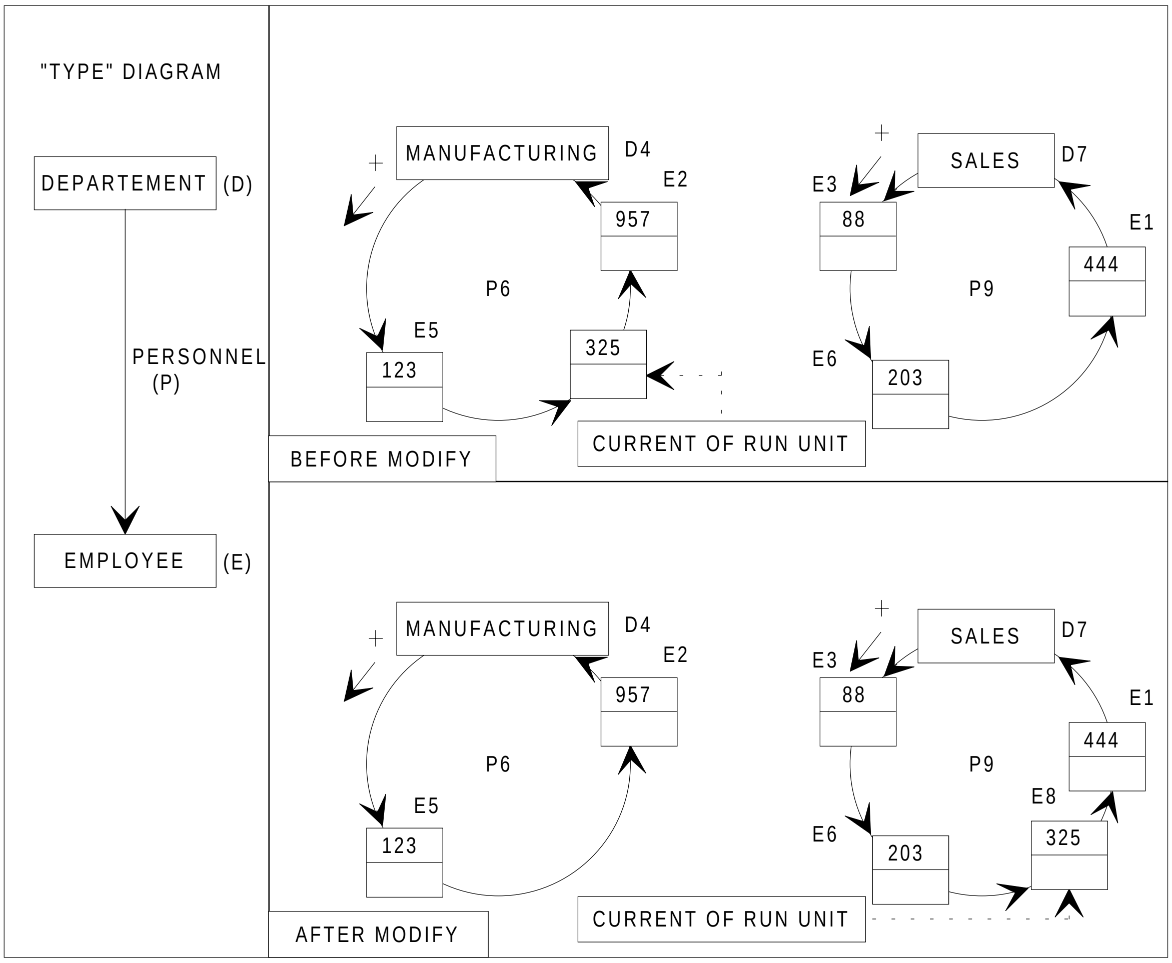


Figure 6-11. Modification of Memberships (Record has no members)

6.2.3.2 Example 2: A Record which is Member of One Set an Owner of One Set

The equivalance, mentioned in the preceding subsection, between a MODIFY statement and an ERASE-STORE sequence, is no longer true when the current-of-run-unit is itself the owner of non-empty set occurrences, since the deletion of the current record would require the deletion of all its members.

The MODIFY statement avoids this chain reaction. It acts upon the current record without affecting its members.

In the data structure of Figure 6-11, the move of employee 325 from the MANUFACTURING department to the SALES department can be described by the same DML sequence as in Example 1. When record E8 changes from set occurrence P6 to set occurrence P9, it brings with it all its SKILL member records. This is a "logical" migration, since no record is physically moved.

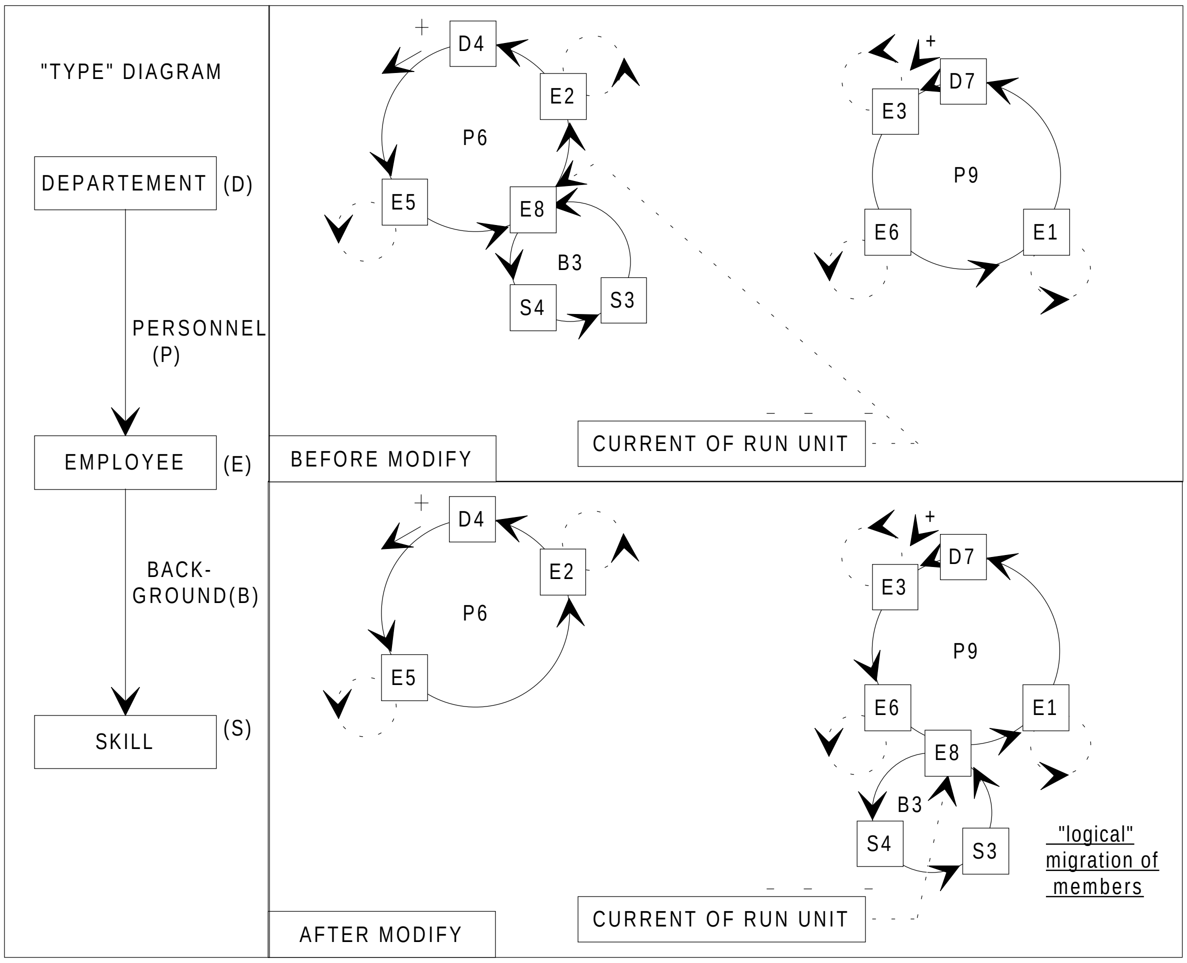


Figure 6-12. Modification of Membership (Record has members)

6.2.3.3 Example 3: A Record which is Member of Two Sets

Let us consider again the data structure composed of CUSTOMER, ORDERS and PART records, in order to understand the case of a record which is member of two sets and changes occurrences of only one set.

Suppose that ORDERS 312 has been entered into the database under CUSTOMER 577 instead of CUSTOMER 597, due to a wrongly punched character in the input data. If the set selection path of S has one level identified by CALC-KEY, the sequence enabling the user to correct the situation could be:

```
MOVE 577 TO C-NUMBER     Prepare selection path for FIND
MOVE 312 TO ORDER-NUMBER
FIND ORDERS WITHIN S USING ORDER-NUMBER
MOVE 597 TO C-NUMBER     Prepare selection path for MODIFY
MODIFY ORDERS ONLY S MEMBERSHIP
```

Figure 6-13, on the following page, shows how ORDERS 312 is logically moved from set occurrence S4 to set occurrence S6, while remaining in set occurrence T2.

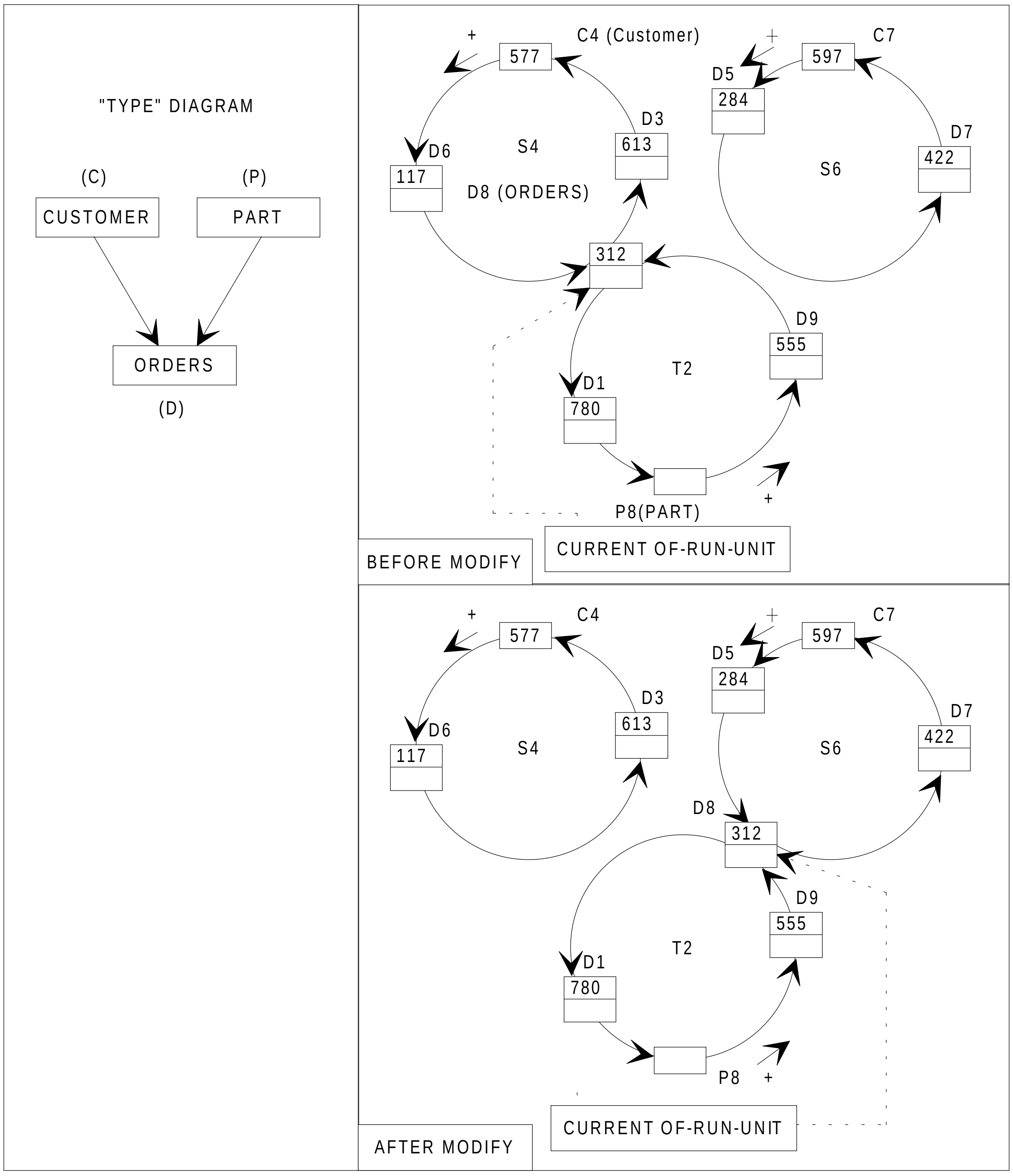


Figure 6-13. Modification of Membership (A Record which is a Member of Two Sets)

6.2.3.4 Example 4: Case of a Set Selection Path Identified by APPLICATION

We will now deal with the problem raised when the selection path of a set specified in the MODIFY statement has only one level, identified by APPLICATION.

The current-of-set-type identifying the new set occurrence must be established prior to retrieving the object record of the MODIFY statement (see Figure 6-14). This retrieval has two conditions:

- it must not rely on a FIND sequence using this current-of-set-type
- it must retain the corresponding "currency".

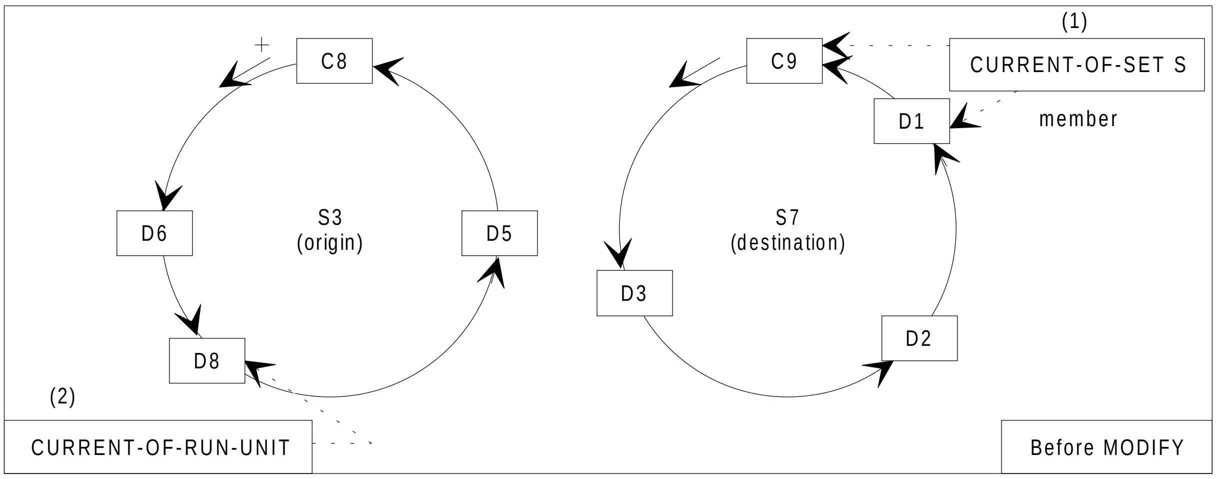


Figure 6-14. Modification of Membership (Set Selection BY APPLICATION)

If the object record has a "DIRECT" or "CALC" LOCATION mode or can be retrieved via another set, the sequence of steps used by the programmer preceding the MODIFY can be:

1) Establish the current-of-set-type of the set occurrence of destination.

2) Retrieve the object record by:

```
   FIND DB-KEY IS ... RETAINING CURRENCY FOR set-name

or FIND ANY record-name RETAINING CURRENCY FOR set-name

or FIND ... WITHIN other-set-name ...
            RETAINING CURRENCY FOR set-name
```

If the object record does not meet the conditions mentioned directly above, the following steps are necessary:

1) Retrieve the object record using any method. Save its data-base-key by an ACCEPT statement or rely on its current-of-record-type if the next step does not modify it.

2) Establish the current-of-set-type of the set occurrence of destination.

3) Retrieve the object record once again by a direct access :

```
   FIND DB-KEY IS SAVE-DBK  RETAINING CURRENCY FOR set-name
or FIND CURRENT record-name RETAINING CURRENCY FOR set-name
```

6.2.4 Modification of the Contents and Membership of a Record

The following statement:

```
       {[record-name]    }           {    ALL         }
MODIFY {                 } INCLUDING {                } MEMBERSHIP
       {{identifier} ... }           { {set-name} ... }
```

is the combination of the MODIFY statements presented in the preceding paragraphs. It allows the following modifications:

- the modification of the contents of some or all of the fields of the current record of the run-unit,
- and the modification of its membership in one or several sets of which it is an AUTOMATIC MANDATORY member or a MANUAL "NOT FIXED" member currently connected.

The conditions of acceptance and the execution of this MODIFY statement are similar to those already described:

1) The modification of a CALC-key may cause a migration

2) The checking of the no-duplicate-control fields related to a given set is performed:

   a) in the same set occurrence, if the set is not mentioned explicitly or implicitly in the INCLUDING list,
   b) in the new set occurrence determined by the set selection criteria if the set is referred to in the INCLUDING list.

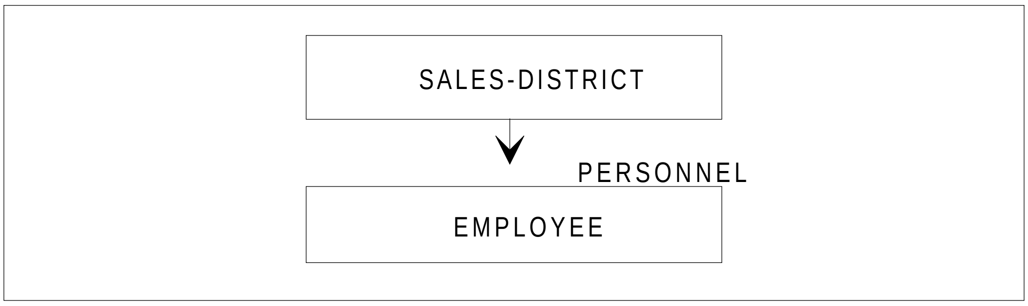
3) The modification of one or several sort-key items related to a given set causes the following:

   a) a reordering of the same set occurrence, if the set is not mentioned explicitly or implicitly in the INCLUDING list
   b) a sorted insertion in the new set occurrence determined by the set selection criteria, if the set is referred to in the INCLUDING list

4) For every set of the INCLUDING list, the record is removed from its current set occurrence and is inserted, according to the set ordering criteria, in the new set occurrence determined by the set selection criteria. The fields involved in the insertion process are those of the record in the database after the modification of of its contents.

6.2.4.1 Example

Consider the following data structure:



and the case of a salesman moving to a new geographical district. His personal address is modified as well as the sales-district to which he belongs.

If the relevant schema clauses are:

```
    RECORD NAME IS SALES-DISTRICT
        LOCATION MODE IS CALC USING SD-NAME DUPLICATES NOT
        WITHIN AREA-1.
        02 SD-NAME TYPE IS CHARACTER 20.
        ...
    RECORD NAME IS EMPLOYEE
        LOCATION MODE IS CALC USING E-NUMBER DUPLICATES NOT
        WITHIN AREA-1.
        02 E-NUMBER TYPE IS SIGNED PACKED DECIMAL 7.
        02 E-ADDRESS TYPE IS CHARACTER 30.
        ...
    SET NAME IS PERSONNEL
        OWNER IS SALES-DISTRICT
        ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS
        DUPLICATES NOT.
    MEMBER IS EMPLOYEE
        ...
        KEY IS ASCENDING E-NUMBER
        SET SELECTION IS
        THRU PERSONNEL OWNER IDENTIFIED BY APPLICATION.
```

then, the DML sequence that reflects the move of employee 8025 from sales district NORTH-FRANCE to sales-district SOUTH-EAST-FRANCE could be written as follows:

```
MOVE "SOUTH-EAST-FRANCE" TO SD-NAME.    Establish the "current-of

FIND ANY SALES-DISTRICT.                set-type" for the new
                                        set occurrence.
MOVE 8025 TO E-NUMBER.                  Establish the "current-of
FIND ANY EMPLOYEE RETAINING PERSONNEL.  run-unit", retaining
                                        the PERSONNEL currency.
MOVE new-address TO E-ADDRESS.          Prepare new address.
MODIFY E-ADDRESS INCLUDING PERSONNEL    Modify contents and
MEMBERSHIP.                             membership.
```

6.2.5 Modification of a Variable Length Record

With a variable length record, one must distinguish **the depending on** control-field from the **controlled field or group**.

Consider the following schema description:

```
                        SCHEMA DDL
RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
01  R01F01  TYPE SIGNED BINARY 15.
01  R01F02  OCCURS 10 DEPENDING ON RO1F01.
 02 R01F03  TYPE CHAR 6.
 02 R01F04  TYPE UNSIGNED PACKED DEC 4,-1.

RECORD NAME IS R02
    LOCATION MODE IS CALC USING R02F01
    WITHIN ANY AREA.
01  R02F01  TYPE SIGNED BINARY 15.
01  R02F02  CHARACTER 20 DEPENDING ON R02F01.

RECORD NAME IS R03
    LOCATION MODE IS CALC USING R03F01
    WITHIN ANY AREA.
01  R03F01  TYPE SIGNED BINARY 15.
01  R03F02  CHARACTER 20 DEPENDING ON R03F01.
```

Fields R01F01, R02F01 and R03F01 are the **depending on control fields.**

Group R01F02 is a **controlled group**, and fields R02F02 and R03F02 are **controlled fields.**

6.2.5.1 Modification of the Record Length

The modification of a **depending on** control field implies a modification of the record length in the database, even though the record is of fixed length in the subschema

Consider the following subschema description against the above schema:

```
                      SUBSCHEMA DDL
RECORD NAME IS R01.

RECORD NAME IS R02.

RECORD NAME IS R03.
01  R03F01  TYPE SIGNED BINARY 15.
```

In this subschema, R01 and R02 are variable length records and R03 is a fixed length record, but the following statement:

```
MODIFY R03F01.
```

is **NOT AUTHORIZED** because it will modify the length of record R03 in the database with an invisible side effect on controlled field R03F02.

Examples of statements that modify the length of records R01 and R02 are given below:

```
MODIFY R01F01.
MODIFY R01F01 R01F03 ( I ).
MODIFY R02.
MODIFY R02F02 R02F01.
```

If the length of a record is modified, there are two possible results:

- record expansion
- record contraction

These are explained in the subsections directly following.

6.2.5.2 Record Expansion

If sufficient room is not available in the page of the current record, the DBCS must perfom a record migration.

If a controlled field is not modified for one of the following reasons:

- either it is not in the subschema
- or it is not in the sub-list of modified items.

The record is padded with **blank characters**.

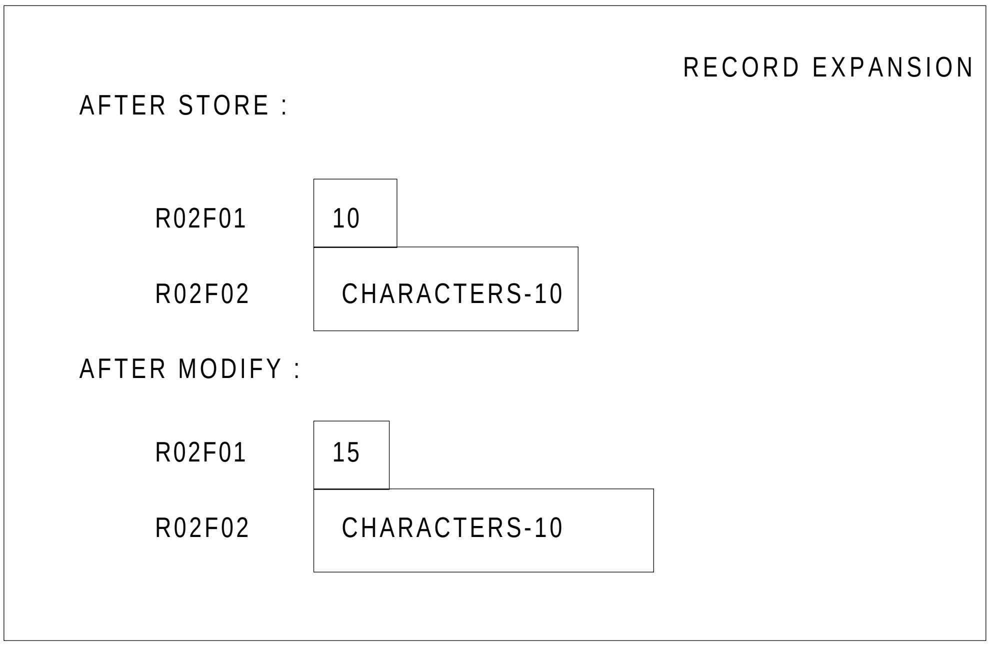
Accordingly, if a non-modified controlled field has a decimal type, since this record expansion takes place, the modification of the record is denied and a data-base-exception occurs.

Examples:

The following sequence:

```
MOVE 10 TO R02FO1.
MOVE "CHARACTERS-10" TO R02F02.
STORE R02.

MOVE 15 TO R02F01.
MODIFY R02F01.
```

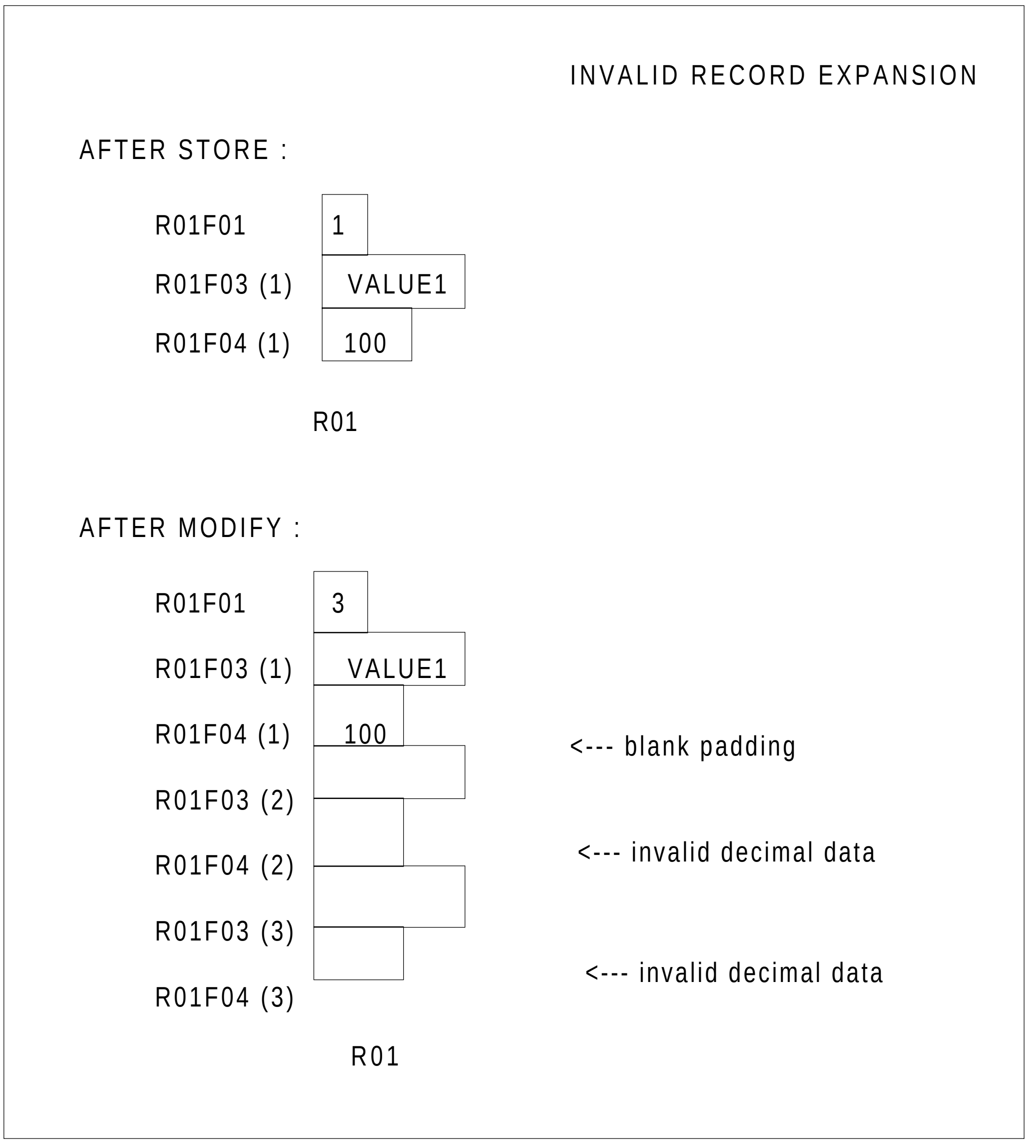
will produce the database record map which is graphically illustrated below:



However the following sequence:

```
MOVE 1  TO R01FO1.
MOVE "VALUE1" TO R01F03 ( 1 ).
MOVE 100 TO R01F04 (1 ).
STORE R01.


MOVE 3 TO R01F01.
MODIFY R01F01.
```

would produce the following invalid database record map:



The modification is therefore is denied.

6.2.5.3 Record Contraction

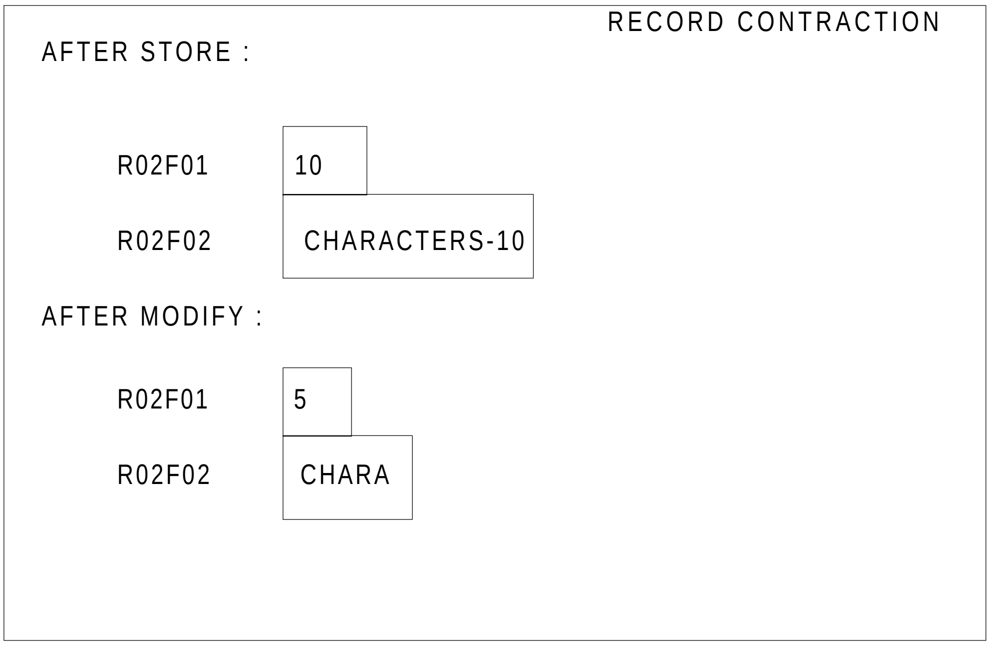
In the event that record contraction results when the record lenght is modified, the database-key of the current record is unchanged.

Examples:

The following sequence:

```
MOVE 10 TO R02FO1.
MOVE "CHARACTERS-10" TO R02F02.
STORE R02.

MOVE 5 TO R02F01.
MODIFY R02F01.
```

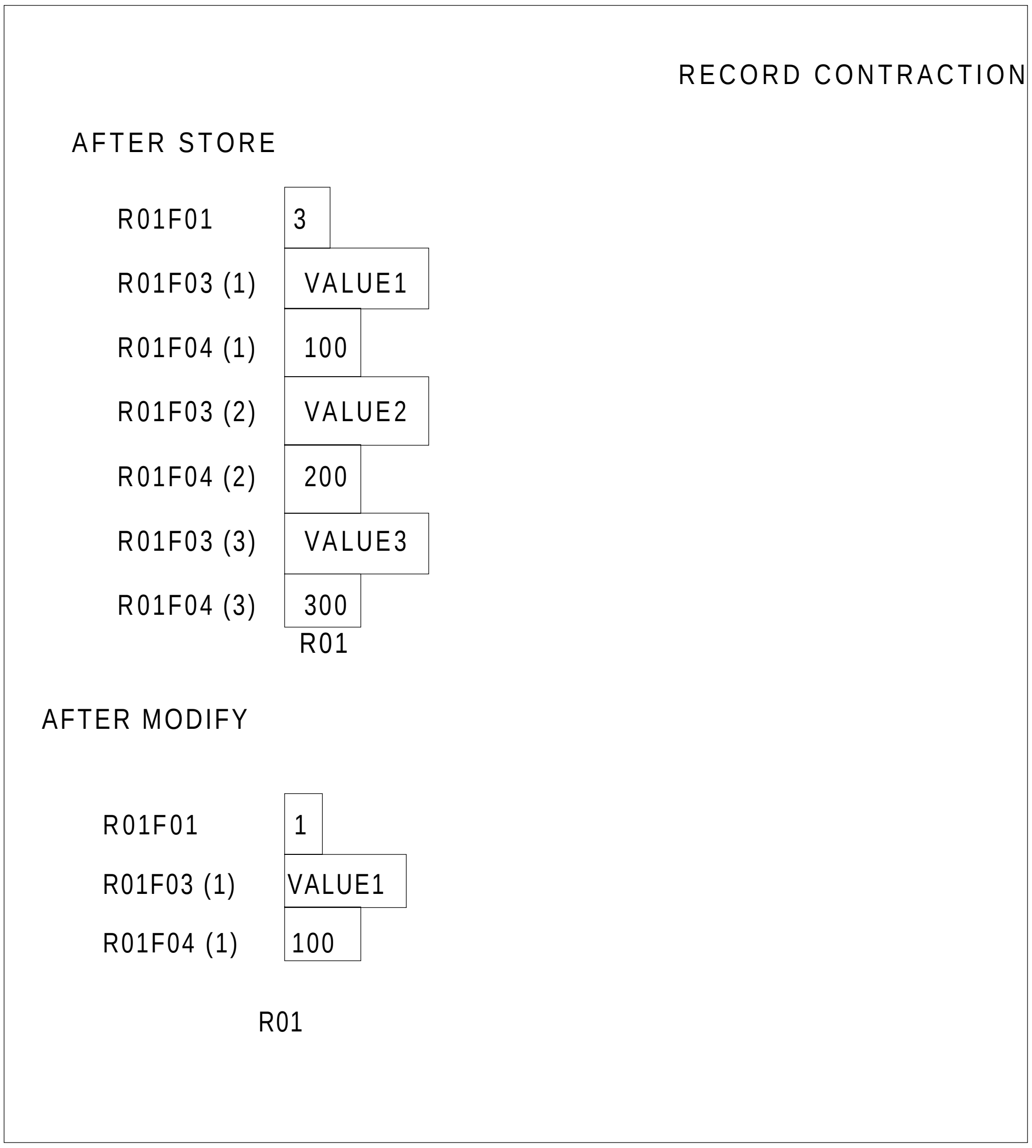
will produce the database record map shown below:



And the following sequence:

```
MOVE 3  TO R01FO1.
MOVE "VALUE1" TO R01F03 ( 1 ).
MOVE 100 TO R01F04 (1 ).
MOVE "VALUE2" TO R01F03 ( 2 ).
MOVE 200 TO R01F04 (2 ).
MOVE "VALUE3" TO R01F03 ( 3 ).
MOVE 300 TO R01F04 (3 ).
STORE R01.


MOVE 1 TO R01F01.
MODIFY R01F01.
```

will produce this database record map:



6.2.5.4 When Migration is Not Allowed

If migration is not allowed, and a migration is a requisite condition for a given MODIFY, the modification is denied and a data-base-exception occurs.

6.2.5.5 When Migration is Allowed

If migration is allowed, and a migration is a requisite condition for a given modification, the record is moved to the next available page of its range.

Since the data-base-key of the record has changed, the following steps are taken:

- All the set pointers which point to the record are updated
- If the data-base-key of the record is a sort-key item in a set, then the set is reordered.
- The currency indicators pointing to the record are updated. This is independent of the presence of the RETAINING clause.

Note that only the modified record migrates and that it cannot migrate outside its area.

If the new data-base-key is called for by the program, an ACCEPT FROM CURRENCY statement must follow the MODIFY statement.

6.2.5.6 Modification of the Controlled Field

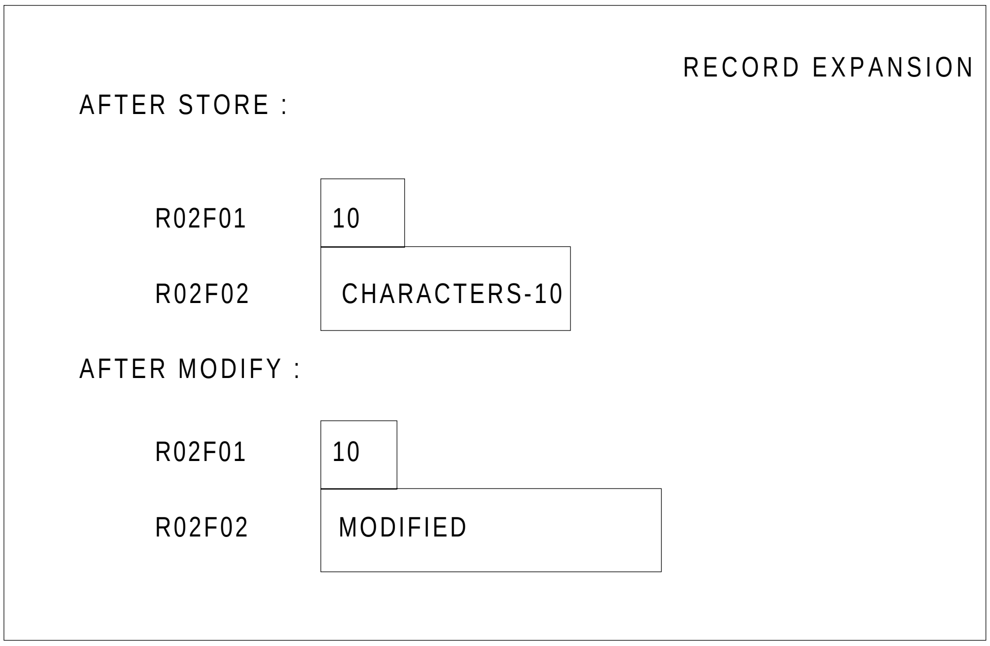
One can modify the controlled field without changing the record length, since the **depending on** control field is not in the sub-list of modified items.

6.2.5.7 Example

The following sequence:

```
MOVE 10 TO R02FO1.
MOVE "CHARACTERS-10" TO R02F02.
STORE R02.

MOVE "MODIFIED" TO R02F02.
MODIFY R02F02.
```

will produce the database record map shown below:



6.2.6 Instances when Modification is Impossible

The MODIFY statement is impossible in certain circumstances. All SET SELECTION Clauses activated by the MODIFY statement must be acceptable for processing by the DBCS. If one of the SET SELECTION Clauses cannot be processed, this is detected either at compile-time if the record type is specified, as in the example below:

```
MODIFY  R01.
```

or at run-time, if the record type has not been specified, as shown in the example below:

```
MODIFY.
```

If the record type has not been specified, the MODIFY statement is denied and a database-exception occurs with the following status code:

DB-STATUS *:* 1104200	THE SUBSCHEMA DOES NOT INCLUDE A DATA ITEM OR
	SET-TYPE REQUIRED BY
	SET SELECTION OR
	SET ORDERING CRITERIA

Three instances where modification is impossible, and the MODIFY statement is consequently denied by the DBCS, are explained in the paragraphs below.

1) If the MODIFY statement implies a modification of record membership in a set, all the **sort-key** items or **duplicates not** items, if any, must be included in the record of the subschema. If such a control field is missing from the subschema, the DBCS cannot perform a record insertion in the new set occurrence and the MODIFY statement is denied. A data-base-exception occurs with codes shown directly above.

Example:

Consider the following Schema DDL:

```
                      SCHEMA DDL
RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
01  R01F01  TYPE CHAR 3.
01  R01F02  TYPE SIGNED BINARY 15.

RECORD NAME IS R02
    LOCATION MODE IS DIRECT DB-PARAM2
    WITHIN ANY AREA.
01  R02F01  TYPE CHAR 8.
01  R02F02  TYPE SIGNED BINARY 15.
01  R02F03  TYPE PACKED DECIMAL 3.

SET NAME IS S01
    OWNER IS R01
    ORDER IS SORTED BY DEFINED KEYS.

MEMBER IS R02
    INSERTION IS AUTOMATIC RETENTION IS MANDATORY
    DUPLICATES ARE NOT ALLOWED FOR R02F01 R02F03
    KEY IS ASCENDING R02F02 R02F03
    DUPLICATES ARE SYSTEM-DEFAULT

    SET SELECTION FOR S01 IS
     THRU S01 OWNER IDENTIFIED BY CALC-KEY.
```

and the corresponding Subschema DDL:

```
                   SUBSCHEMA DDL
RECORD NAME IS R01.

RECORD NAME IS R02.
01  R02F01  TYPE CHAR 8.
01  R02F02  TYPE SIGNED BINARY 15.

SET NAME IS S01.
```

The Subschema DDL excludes control field R02F03. Therefore, the following statements are impossible:

```
MODIFY R02F01.   (R02F03 missing in
                  DUP NOT clause   )
MODIFY R02F02.   (R02F03 missing in
                  SORT-KEY clause  )
MODIFY R02.      (both preceding reasons)
```

2) If the MODIFY statement implies a modification of a data item which participates in a CHECK Clause, all the items of this CHECK Clause must be included in the subschema. If such a control field is missing from the subschema, the DBCS cannot perform the CHECK Clause and the MODIFY statement is denied. A data-base-exception occurs containing the status code indicated at the beginning of this subsection.

Example:

Consider the following Schema DDL:

```
                   SCHEMA DDL
RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
    CHECK R01F02 > R01F01.
01  R01F01  TYPE SIGNED BINARY 15.
01  R01F02  TYPE SIGNED BINARY 15.
```

and the corresponding Subschema DDL which contains no entry for control field R01F01:

```
                   SUBSCHEMA DDL
RECORD NAME IS R01.
01  R01F02  TYPE SIGNED BINARY 15.
```

Without that entry, the following statements are impossible:

```
MODIFY R01F02.   (R01F01 missing in
                  CHECK   clause   )
MODIFY R01.      (same reason      )
```

3) If the MODIFY statement implies a modification of a data item which participates in a secondary key, the secondary key must be included in the subschema. If such a control field is missing from the subschema, the DBCS cannot perform an index modification and the MODIFY statement is denied. A data-base-exception occurs containing the status code indicated earlier in this subsection.

Example:

Consider the following Schema DDL:

```
                   SCHEMA DDL
RECORD NAME IS R01
    LOCATION MODE IS CALC USING R01F01
    WITHIN ANY AREA.
    KEY NAME IS K01 USING ASCENDING R01F01 R01F02.
01  R01F01  TYPE SIGNED BINARY 15.
01  R01F02  TYPE SIGNED BINARY 15.

KEY NAME IS K01
    DUPLICATES ARE ALLOWED.
```

and the corresponding Subschema DDL which contains no entry for control field R01F01:

```
                 SUBSCHEMA DDL
RECORD NAME IS R01.
01  R01F02  TYPE SIGNED BINARY 15.
```

Without that entry, the following statements are impossible:

```
MODIFY R01F02.   (R01F01 missing in K01)

MODIFY R01.      (same reason)
```

6.2.7 With a Detachable Index

If a SET SELECTION Clause activated by the MODIFY statement invokes a secondary key which is declared in a DETACHABLE INDEX, the DBCS performs a consistency check on the index which compares the data-base-key of the selected record which is found via the index retrieval path to the database record. If the data-base-key found leads to a missing record in the database, or to a different record type, the MODIFY statement is denied and a data-base-exception occurs with the following status code:

```
DB-STATUS:   1173615       ( INDEX INCONSISTENCY )
```

If a field which participates in a secondary key is modified, the secondary key entry is updated in the corresponding index, if it is **an attached** index.

If an index is **detachable** the DBCS will not be able to find a secondary key entry there, and therefore the index has been left in an inconsistent state with regard to the database. In that case, the MODIFY statement is denied and a data-base-exception occurs containing the status code: 1173615.

6.3 ERASE

6.3.1 General

The ERASE function **removes records from the database**.

When the removal is restricted to one record (called a simple ERASE), the objet record of the ERASE statement is the current-of-run-unit.

When the removal is extended to several records (called a multiple ERASE), the object records are:

- the current-of-run-unit
- all members of the sets of which the current-of-run-unit is the owner
- and, iteratively, all the members of these members.

6.3.2 Simple ERASE

The following statement:

```
ERASE [record-name]
```

removes **one record** or **several records**, in the case of FIXED members, from the database.

When **record-name** is specified, the DBCS checks that the current-of-run-unit is of this type.

The statement is not executed if the current-of-run-unit is currently the owner of at least one non-empty set occurrence with member records which have an OPTIONAL or MANDATORY membership. In other words, all the OPTIONAL or MANDATORY members of a record must be erased (or disconnected, if applicable) before the object record itself can be erased.

The execution of the statement involves the three operations listed below:

1) The disconnection of the record from all the sets of which it is an actual member (AUTOMATIC MANDATORY member, MANUAL OPTIONAL member or FIXED member currently connected).

2) The deletion of the record; the space that the record occupied in the database and the corresponding data-base-key consequently become free.

3) The deletion of all FIXED members of sets for which the object record is currently the owner. Any record deleted in this way is treated as though it were the object of a simple ERASE statement. The process is repeated until all hierarchically related records have been removed from the database. This case is a form of the multiple ERASE which is described in the following subsection.

A successful ERASE statement has the following consequences:

1) The current-of-run-unit is nulled.

2) If the record erased is the current of its record-type, the corresponding, current-of-record-type is nulled.

3) If the record erased is the current of a set-type of which it is the owner, the corresponding current-of-set-type is nulled. (In this situation, the entire set occurrence has disappeared.)

4) If the record erased is the current of a set-type of which it is a member, the corresponding current-of-set-type is updated to identify the relative position within the set of the erased record. This relative position is a virtual pointer: it does not point to a record, but memorizes the data-base-keys of the NEXT, PRIOR, and OWNER records of the erased record. This enables the program to use this virtual current-of-set-type in subsequent DML statements such as:

```
FIND NEXT (PRIOR,OWNER,FIRST,LAST,integer) WITHIN set-name.
```

   The only FIND ... WITHIN set-name statement which is prohibited is the "FIND CURRENT WITHIN set-name" option, since it attempts to retrieve the erased record:

5) If the record erased is the current of a key-type, the corresponding current-of-key-type is updated to identify the relative position within the secondary key of the erased record. This relative position is a virtual pointer: it does not point to a record, but memorizes the data-base-keys of the NEXT and PRIOR records of the erased record within the secondary key index. This enables the program to use this virtual current-of-key-type in subsequent DML statements such as:

```
FIND NEXT (PRIOR,FIRST,LAST,integer) WITHIN key-name.
```

In this case, the only FIND... WITHIN key-name statement which is prohibited is the "FIND CURRENT WITHIN key-name" since it attempts to retrieve the erased record.-

6) If the record erased is the current of its realm, the corresponding current-of-realm is updated to identify the relative position within the realm of the erased record. This relative position, again, is a **virtual pointer**, since it does not point to a record, but it can be referenced in subsequent DML statements such as:

```
FIND NEXT (PRIOR) WITHIN realm-name.
```

   However the "FIND CURRENT WITHIN realm-name" statement is prohibited since it attempts to retrieve the erased record.

Figures 6-15 and 6-16, on the following page, illustrate the concept of a virtual pointer.

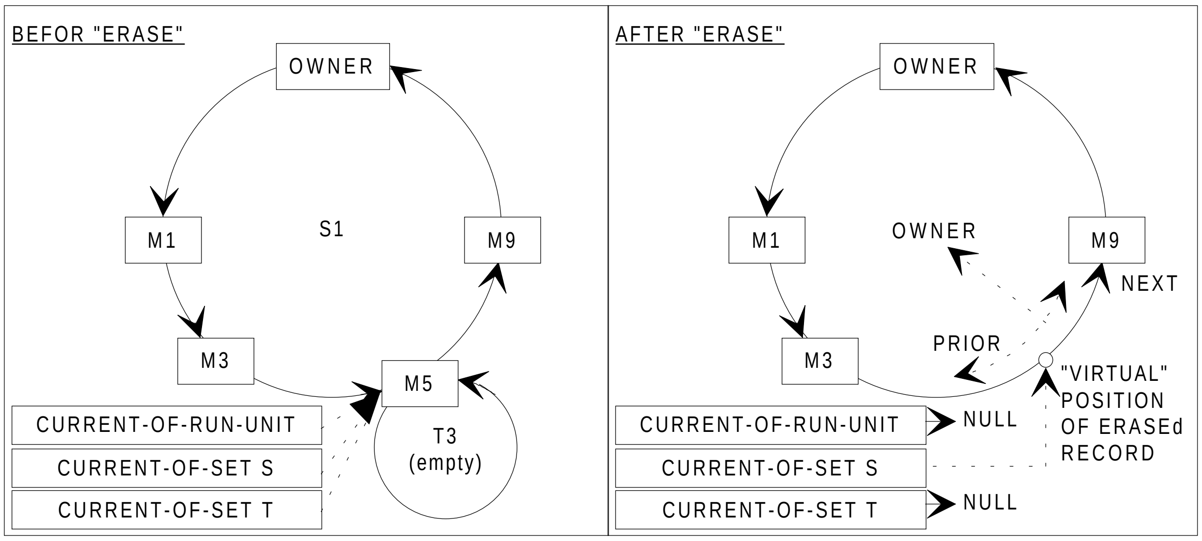


Figure 6-15. Virtual Current-of-set-type after an ERASE statement

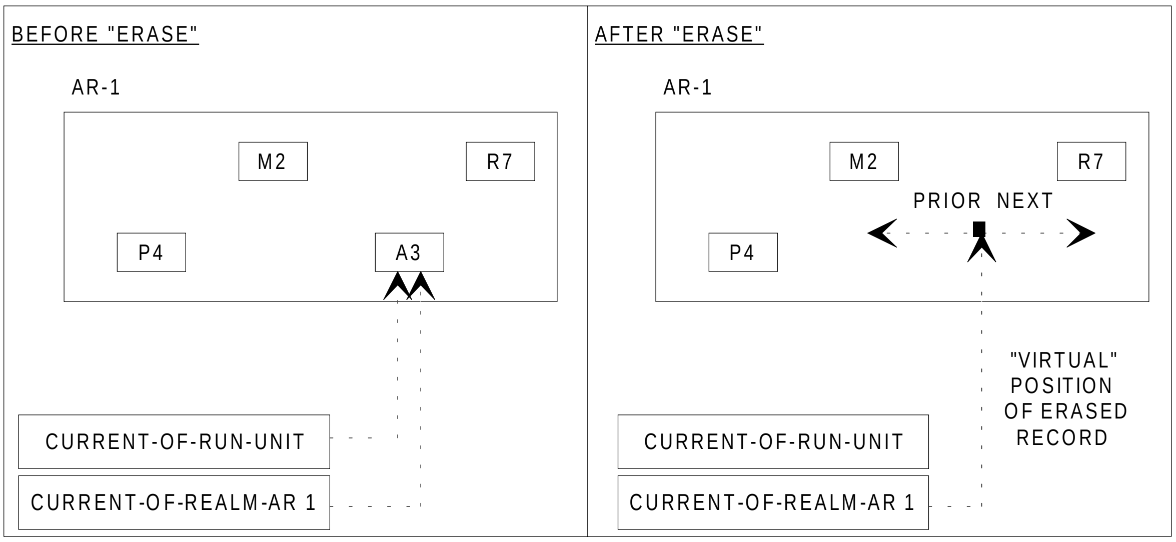


Figure 6-16. Virtual Current-of-realm after an ERASE statement

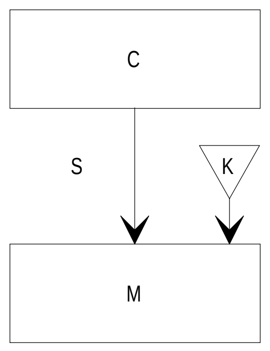
6.3.2.1 Example 1

Consider the data structure of Figure 6-17, further on in this subsection, in terms of the following problem: you must search area AR-1 for every C record and then erase the C record found, after having erased its M members, if there are any. Prior to erasing an M record, you must display its contents.

The deletion sequence might be:

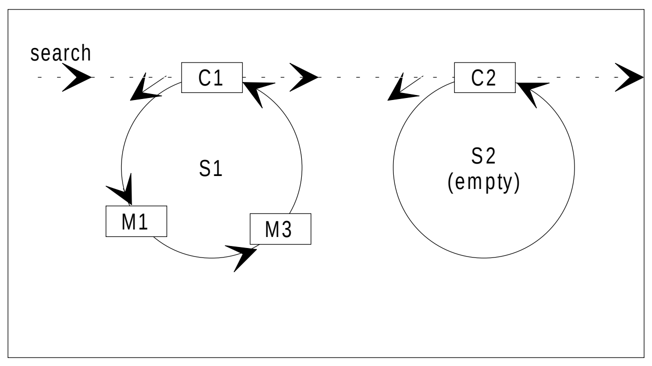
```
INIT-0-LOOP.  FIND FIRST C WITHIN AR-1.     Locate first C record
        GO TO TEST-EOR.                     in area.
OWNER-LOOP.   FIND NEXT C WITHIN AR-1.      Locate next C record
                                            in area.
TEST-EOR.     IF DB-STATUS = "0502100"      Test "end-of-realm"
                                            condition.
              GO TO END-0-LOOP.
        IF S OWNER GO TO                    Test if record C has
        MEMBER-LOOP.                        any members.
OWNER-ERASE.  ERASE C.                      Delete record C.
        GO TO OWNER-LOOP.
MEMBER-LOOP.  FIND NEXT M WITHIN S.         Locate next M record in S.
        IF DB-STATUS = "0502100"            Test "end-of-set"
        GO TO END-M-LOOP.                   condition.
        GET M.                              Get record M
        DISPLAY ...                         for display .
        ERASE M.                            Delete record M.
        GO TO MEMBER-LOOP.
END-M-LOOP.   FIND OWNER WITHIN S.          Re-establish record C as
        GO TO OWNER-ERASE.                  "current-of-run-unit"
END-O-LOOP. ...                             For ERASE
```

"TYPE" DIAGRAM



"OCCURRENCE" DIAGRAM

area AR-1



"CURRENCY" CHART

The symbol "-" means that the currency is not altered by the DML statement

DML STATEMENT	CURRENT OF RUN UNIT	CURRENT OF REALM AR-1	CURRENT OF RECORD C	CURRENT OF RECORD M	CURRENT OF SET S	CURRENT OF KEY K
FIND FIRST C WITHIN AR-1	C1	C1	C1	-	C1	-
FIND NEXT M WITHIN S	M1	M1	-	M1	M1	M1
ERASE M	$\underline{\text{Null}}$	$\boxed{\text{Virtual}}$	-	$\underline{\text{Null}}$	$\boxed{\text{Virtual}}$	$\boxed{\text{Virtual}}$
FIND NEXT M WITHIN S	M3	M3	-	M3	M3	M3
ERASE M	$\underline{\text{Null}}$	$\boxed{\text{Virtual}}$	-	$\underline{\text{Null}}$	$\boxed{\text{Virtual}}$	$\boxed{\text{Virtual}}$
FIND NEXT M WITHIN S (EOS)	-	-	-	-	-	-
FIND OWNER WITHIN S	C1	C1	C1	-	C1	-
ERASE C	$\underline{\text{Null}}$	$\boxed{\text{Virtual}}$	$\underline{\text{Null}}$	-	$\underline{\text{Null}}$	-
FIND NEXT C WITHIN AR-1	C2	C2	C2	-	C2	-
ERASE C	$\underline{\text{Null}}$	$\boxed{\text{Virtual}}$	$\underline{\text{Null}}$	-	$\underline{\text{Null}}$	-
FIND NEXT C WITHIN AR-1(EOS)	-	-	-	-	-	-

Figure 6-17. The Simple ERASE Sequence

The currency chart in Figure 6-17 shows the evolution of currency indicators during processing. It also illustrates the virtual current-of-set-type and virtual current-of-realm. GET statements are not represented in the figure because they do not affect currency indicators. The effects of the erase sequence are further explained below:

FIND FIRST C WITHIN AR-1 retrieves C1.

IF S OWNER indicates that there are some members in S occurrence S1.

FIND NEXT M WITHIN S retrieves M1. M1 becomes the current-of-key because key K is declared on M.

ERASE M deletes M1. The current-of-run-unit is nulled. Since M1 was the current of its record-type, the "current-of-record M" is nulled. Since M1 was the "current-of-set S" (current member), the "current-of-set S" becomes virtual and points to a space between C1 and M3. Because M1 was the "current-of-key K", the "current-of-key K" becomes virtual and points to the next record in key K. Because M1 was the "current of AR-1", the "current-of-realm AR-1" becomes virtual.

FIND NEXT M WITHIN S begins a search from the virtual "current-of-set S" and retrieves M3.

ERASE M deletes M3. The currency updating is the same as for M1.

FIND OWNER WITHIN S starts from the virtual "current-of-set S" and retrieves C1.

ERASE C deletes C1. The "current-of-run-unit" and "current-of-record C" are nulled. As C1 was current of S (as owner), the "current-of-set S" is nulled (not made virtual). As C1 was current of AR-1, the "current-of-realm AR-1" becomes virtual.

FIND NEXT C WITHIN AR-1 starts from the virtual "current-of-realm AR-1" and retrieves C2.

IF S OWNER indicates that set occurrence S2 is empty, therefore C2 can be erased. The processing goes on until an "end-of-realm" condition is returned.

6.3.2.2 Example 2

Consider set S01, illustrated in Figure 6-18, which is located between C and fixed member M. Suppose that record C1 is the current-of-run-unit. The Simple "ERASE C" statement will delete C1 and all members of that set (M1, M2 and M3).

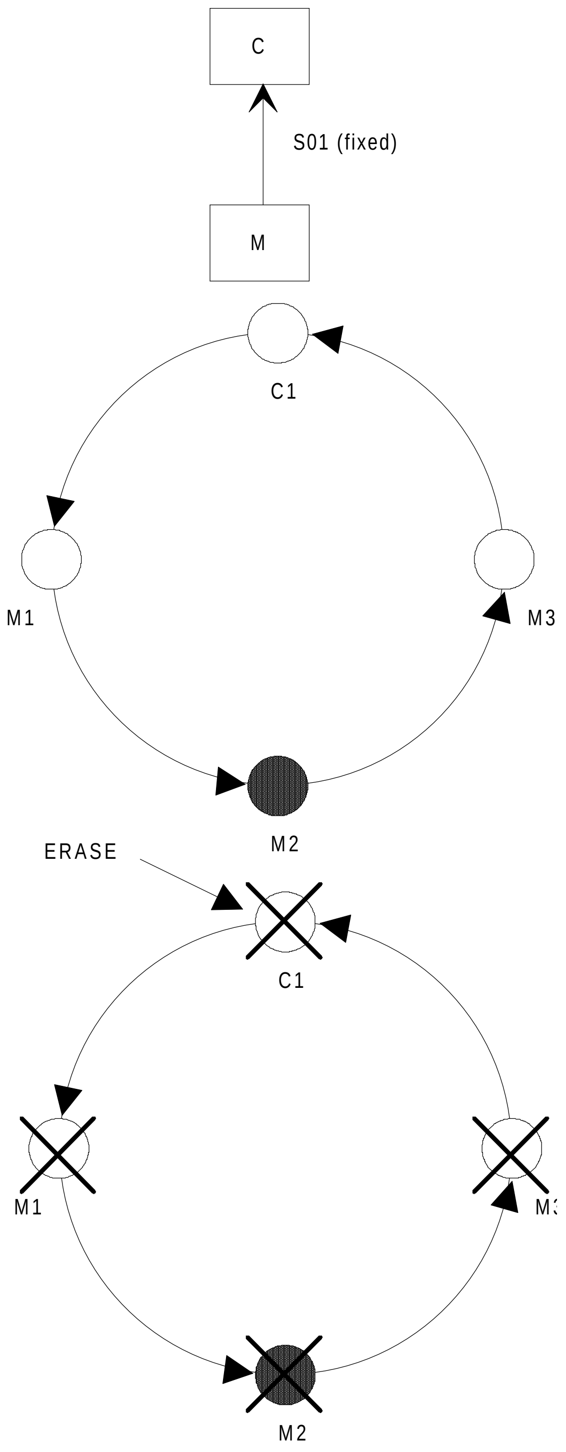
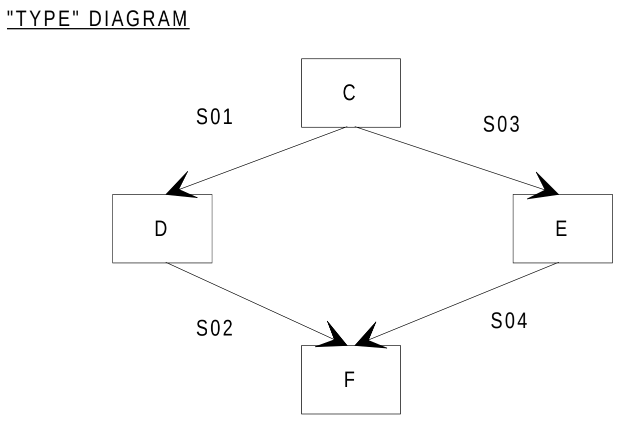


Figure 6-18. Simple ERASE With Fixed Members

- Now consider the following complex network:



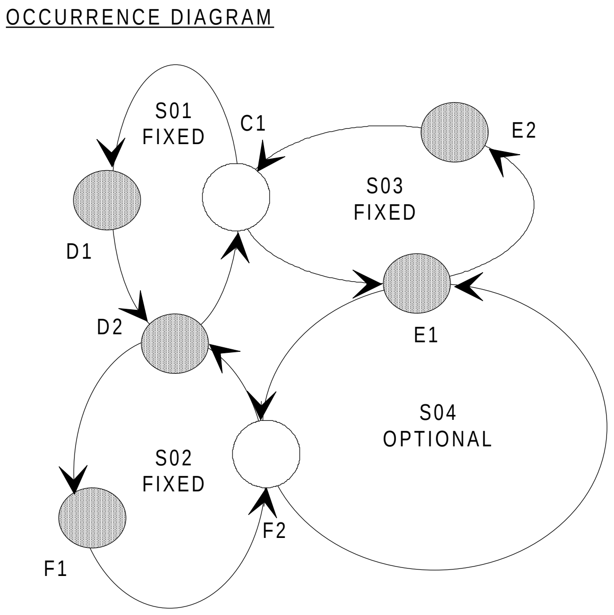


Figure 6-19. Simple ERASE With a Complex Network

In this network, two cases must be distinguished. The first case, illustrated in Figure 6-20, which describes set S01 which was declared **before** set S02, and the second, illustrated in Figure 6-21, which describes another set S01, which was declared after set S02.

CASE 1:

Set **S01**, which was declared **before** set S02 in the Schema DDL (Figure 6-20):

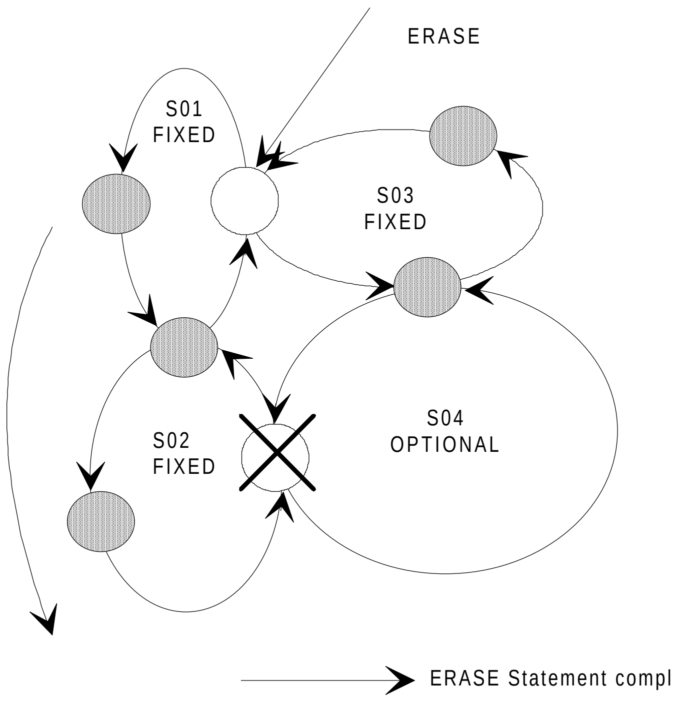


Figure 6-20. Successful Simple ERASE With a complex Network

In this case, the "ERASE C" statement will delete the following records, **beginning with the members of set** S01, in that order:

- D1 (fixed member)
- D2 (fixed member)

Since D2 is the owner of a non-empty set occurrence, S02, the ERASE process is applied recursively to D2 and the following records are deleted:

- F1 (fixed member)
- F2 (fixed member)

Next, the ERASE statement acts on **Set S03**, and the following records are deleted:

- E1 (fixed member, owner of an empty set)
- E2 (fixed member)

The process ends by deleting C1.

At this point, **the ERASE statement is successfully completed.**

CASE 2:

Set S01, which was declared **after** set S02 in the Schema DDL (Figure 6-21):

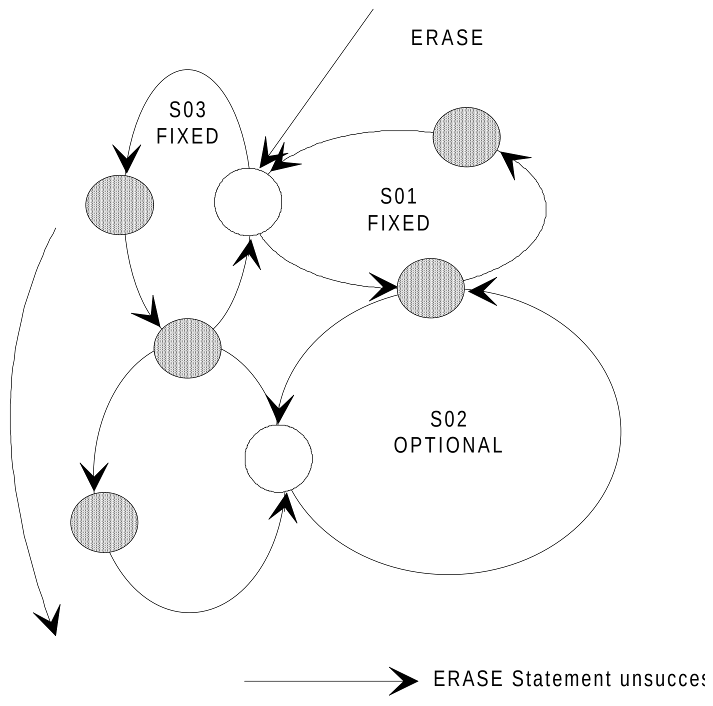


Figure 6-21. Unseccessful Simple ERASE With a Complex Netword

In this case, the "ERASE C" statement will attempt to delete record E1 **first**. But this record cannot be deleted by a simple ERASE statement because **it has at least one non-fixed member**, **which is F2.** For this reason, the ERASE statement is denied and a data-base-exception occurs.

6.3.3 Multiple ERASE

6.3.3.1 ERASE ALL MEMBERS Statement

The following statement:

```
ERASE [record-name] ALL MEMBERS
```

removes one or more records from the database.

If **record-name** is specified, the DBCS checks that the current-of-run-unit is of this type. The current record of the run-unit is first removed as in the case of a simple ERASE. Then the removal function is applied to all the members, if any, of the set occurrences hierarchically subordinate to the current record of the run-unit.
For example, given the data structure in Figure 6-22, an ERASE ALL applied to record A will delete:

- record A
- the B members of the occurrence S whose owner was A
- the D members of the occurrence U whose owner was A (after D is disconnected from its occurrence W)
- then, the C members of the occurrence T related to each deleted B member are deleted.
- and last, the E and F members of the occurrence V related to each deleted D member are deleted (after F is disconnected from its occurrence X, if it is currently connected)

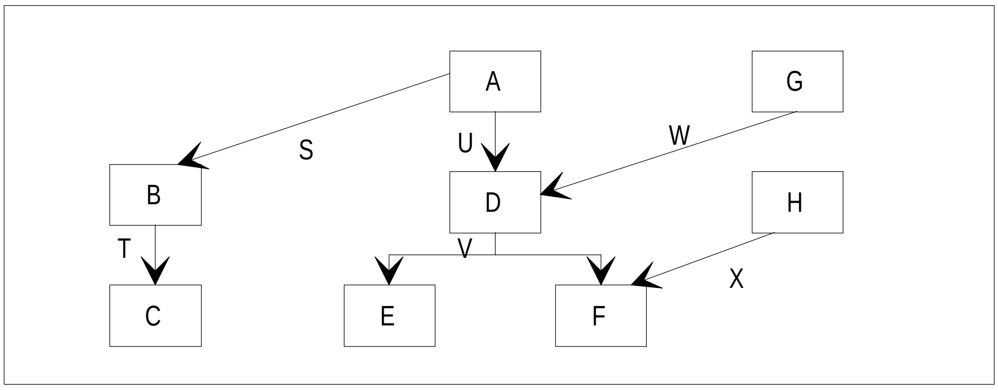


Figure 6-22. Example of a Multiple ERASE

When the ERASE ALL statement has executed successfully, it has the following consequences:

- The current-of-run-unit is nulled.
- If any erased record is the current record of its record-type, the corresponding current-of-record-type is nulled.
- If any erased record is the current record of a set-type hierarchically subordinate to the current record of the run-unit, the corresponding current-of-set-type is nulled (since the whole set occurrence has disappeared).
- If any erased record is the current record of a set-type not hierarchically subordinate to the current record of the run-unit, the corresponding current-of-set-type is made virtual.
- If any erased record is the current record of its realm, the corresponding current-of-realm becomes virtual.

6.3.3.2 Example

Consider again the example in Figure 6-17, but using the ERASE ALL function. If we assume that it is no longer required to retrieve the M records for display, the DML sequence becomes:

```
INIT-O-LOOP.  FIND FIRST C WITHIN AR-1.    Locate first C record
        GO TO TEST-EOR.                    in area.
OWNER-LOOP.   FIND NEXT C WITHIN AR-1.     Locate next C record
                                           in area.
        IF DB-STATUS = "0502100"           Test "end-of-realm"
        GO TO END-O-LOOP.                  condition.
        ERASE C ALL.                       Erase C and its members.
        GO TO OWNER-LOOP.
END-O-LOOP.     ...
```

6.3.3.3 ERASE PERMANENT MEMBERS Statement

The following statement:

```
ERASE [record-name] PERMANENT MEMBERS
```

removes one or more records from the database.

If **record-name** is specified, the DBCS checks that the current-of-run-unit is of this type.

The current record of the run-unit is first removed as in the case of a simple ERASE. Then, the removal function is applied to all **permanent** (not optional) members, if any, of set occurrences whose owner is the current record of the run-unit. The optional members of these set occurrences are only **disconnected**

The process is repeated on all **removed** records until all hierarchically related records have been processed.

6.3.3.4 Example

Consider the example in Figure 6-23, on the following page:

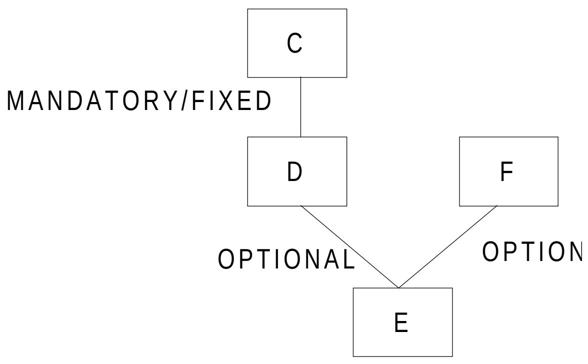


Figure 6-23. Example of the ERASE PERMANENT Statement

Two examples of the statement "ERASE C PERMANENT" when C1 is the current-of-run-unit will be illustrated in Figures 6-24 and 6-25 reslpectively, which appear on the two following pages:

- in a hierarchy
- in a network

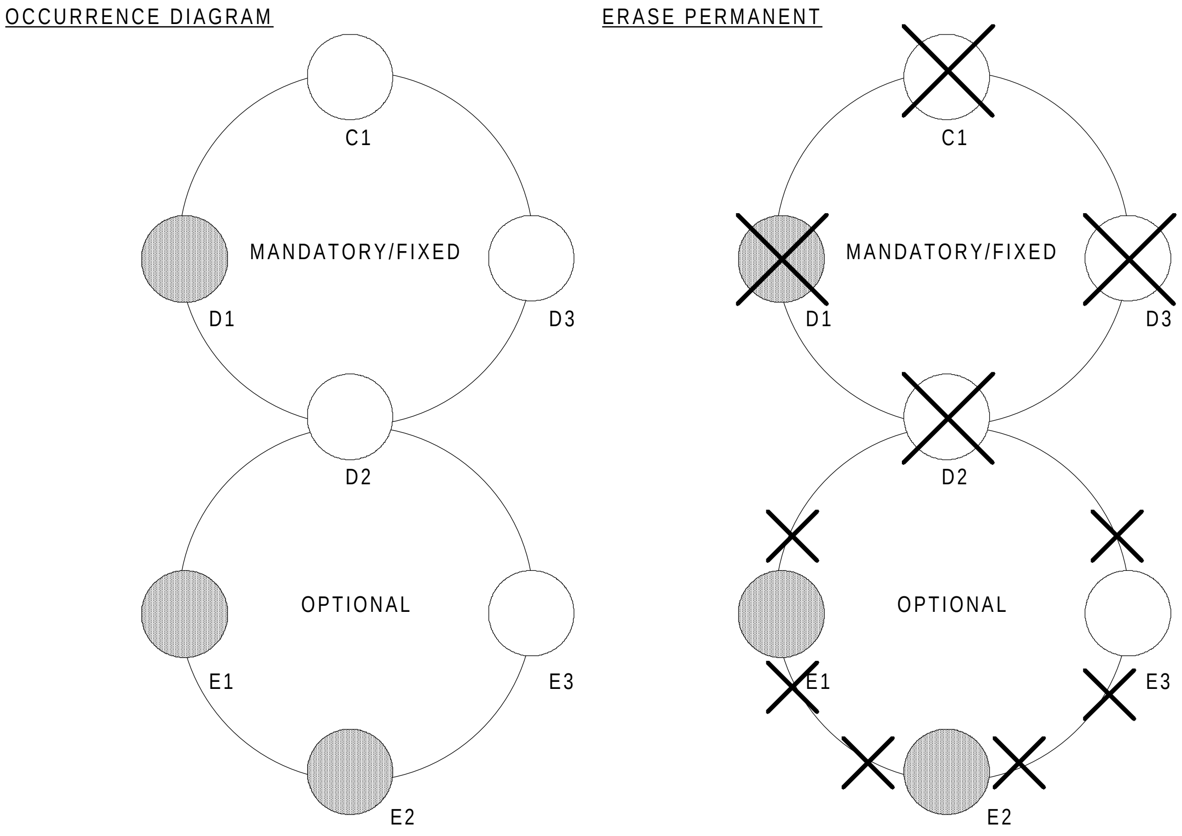


Figure 6-24. ERASE PERMANENT applied to a Hierarchical Occurence

When the set occurrences are organized in a hierarchy, the "ERASE C PERMANENT" has the following effects:

- FIXED members D1, D2 and D3 are **deleted**
- OPTIONAL members E1, E2 and E3 are **only disconnected**

When the set occurrences are organized in a network, the "ERASE C PERMANENT" statement has the following effects (Figure 6-25):

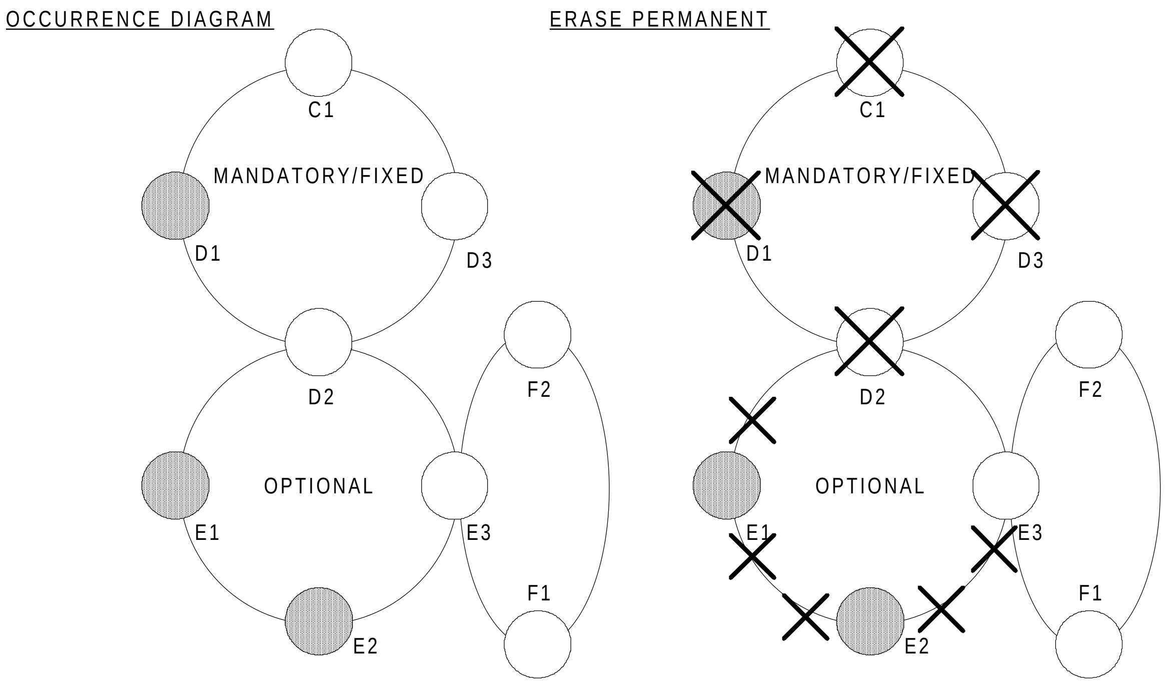


Figure 6-25. ERASE PERMANENT with a Network Occurence

- FIXED members D1, D2 and D3 are **deleted**
- OPTIONAL members E1, E2 and E3 are **only disconnected** as in a hierarchical structure, **even if record E3 is a member of another set occurrence.**

6.3.3.5 ERASE SELECTIVE MEMBERS Statement

The following statement:

```
ERASE [record-name] SELECTIVE MEMBERS
```

removes one or more records from the database.

If record-name is specified, the DBCS checks that the current-of-run-unit is of this type.

The current record of the run-unit is first to be removed as with the PERMANENT ERASE statement, which means that the removal function is applied to all the permanent (not optional) members, if any, of the set occurrences of which the current record of the run-unit is owner. The optional members of these set occurrences are either:

- only **disconnected**, if they are members of at least one other set occurrence
- **removed**, if they are not members of another set occurrence

The process is repeated on all **removed** records until all hierarchically related records have been processed.

6.3.3.6 Example

Two examples of the "ERASE C SELECTIVE" statement when C1 is the current-of-run-unit will be illustrated with the examples of ERASE PERMANENT which have already been illustrated in Figures 6.24 and 6.25.

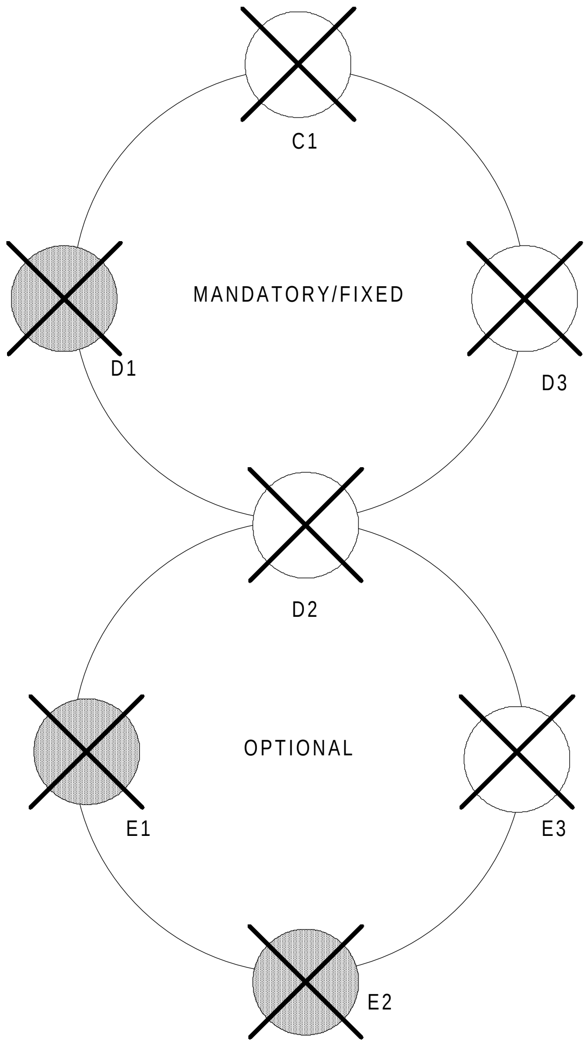


Figure 6-26. ERASE SELECTIVE with a Hierarchical Occurence

When the set occurrences are organized in a hierarchy, as shown in Figure 6-26, the ERASE SELECTIVE statement has the following effects:

- FIXED members D1, D2 and D3 are **deleted**
- OPTIONAL members E1, E2 and E3 are also **deleted because they are not member of another set occurrence.**

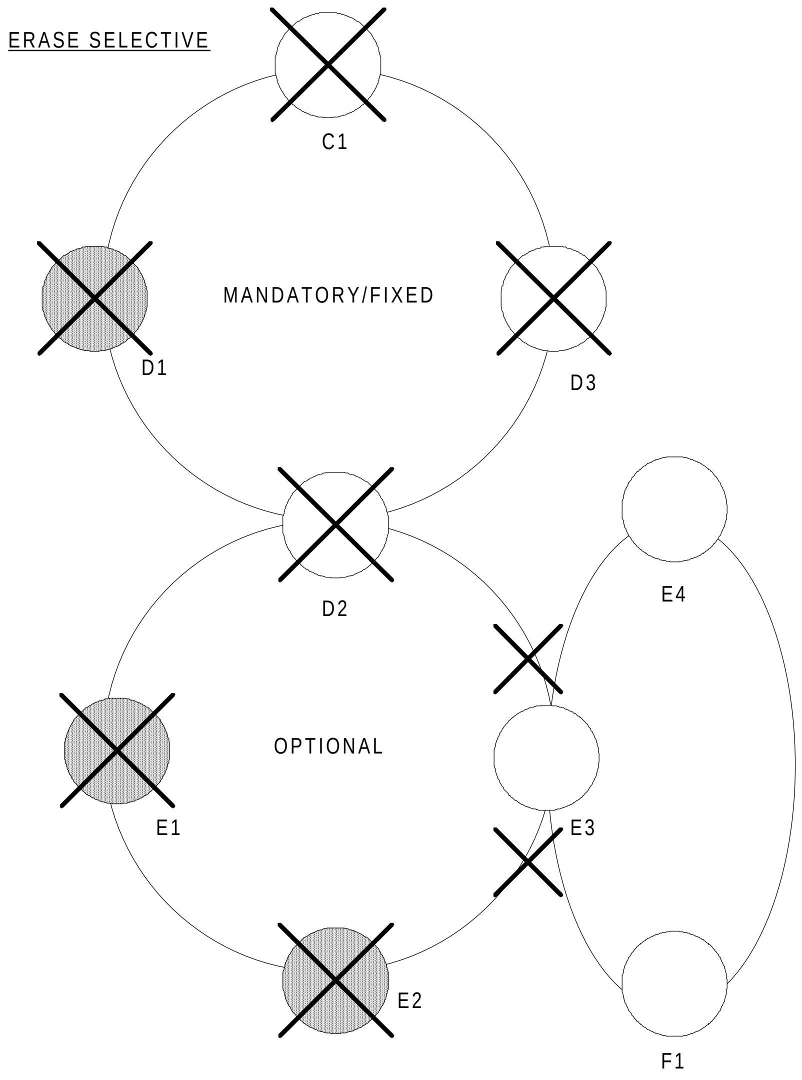


Figure 6-27. ERASE SELECTIVE with a Network Occurrence

When the set occurrences are organized in a network, as shown in Figure 6-27, the ERASE SELECTIVE statement has the following results:

- FIXED members D1, D2 and D3 are **deleted**
- OPTIONAL members E1 and E2 are **deleted** as in the preceding example
- OPTIONAL member E3 is only **disconnected because it is a member of another set occurrence**.

6.3.4 The Case when ERASE is Impossible

The ERASE statement is impossible on a record if at least one secondary key declared on it is missing in the subschema.

The DBCS detects that the statement is impossible either at compile-time if the record type is specified, as in the example below:

```
ERASE   R01.
```

or at run-time if the record type is not specified, as in the following example:

```
ERASE.
```

In these two cases, the ERASE statement is denied and a database exception occurs with the following status code:

```
DB-STATUS : 0404200
          ( THE SUBSCHEMA DOES NOT INCLUDE A DATA ITEM OR
          SET-TYPE REQUIRED BY SET SELECTION OR SET ORDERING
          CRITERIA )
```

6.3.5 The ERASE Statement with a Detachable Index

During the process of erasing a record, all the secondary key entries which are declared on this record in the Schema DDL are removed from the corresponding indexes, if these indexes are **attached**.

If an index is **detachable** the DBCS will be unable to locate a secondary key entry in the index, which means that the index has been left in an inconsistent state with regard to the database. If this is the case, the ERASE statement is denied, and a data-base-exception occurs with the following status code:

```
    DB-STATUS :  0473615       ( INDEX INCONSISTENCY )
```

6.4 CONNECT

6.4.1 General

The CONNECT function **causes a record, which is already stored in the database, to become an actual member of a set in which it has been declared as a MANUAL member.**

Its format of the CONNECT statement is the following:

```
CONNECT [record-name] TO set-name
```

The object record of the statement is the current-of-run-unit. If **record-name** is specified, the DBCS checks that the current-of-run-unit is of this record-type.

The object record must be declared in the schema as a MANUAL member of the set-type specified by set-name. It must not be currently connected to an occurrence of this set-type.

The set occurrence to which the record will be connected is defined by the set selection criteria of the specified set. The parameters of the set selection path must be prepared beforehand in the UWA.

If the set is the one specified in the VIA phrase of the LOCATION Clause of the record and if the record is located WITHIN AREA OF OWNER, the DBCS checks that the owner of the selected set occurrence is in the same area as the current record of the run-unit. If not, a data-base-exception occurs.

The point of insertion into the set occurrence is defined by the **set ordering criteria**. If sort-keys or no-duplicate-control-fields are involved in the insertion process, their values are those of the record in the database; the UWA record is not considered.

Following the successful completion of a CONNECT statement, provided that the RETAINING phrase is omitted, the record becomes the current-of-set-type of the specified set. The other currency indicators remain unchanged.

Figure 6-28, on the following page, illustrates the action of the CONNECT statement when the set selection path has o nly one level identified BY APPLICATION and the set order is NEXT.

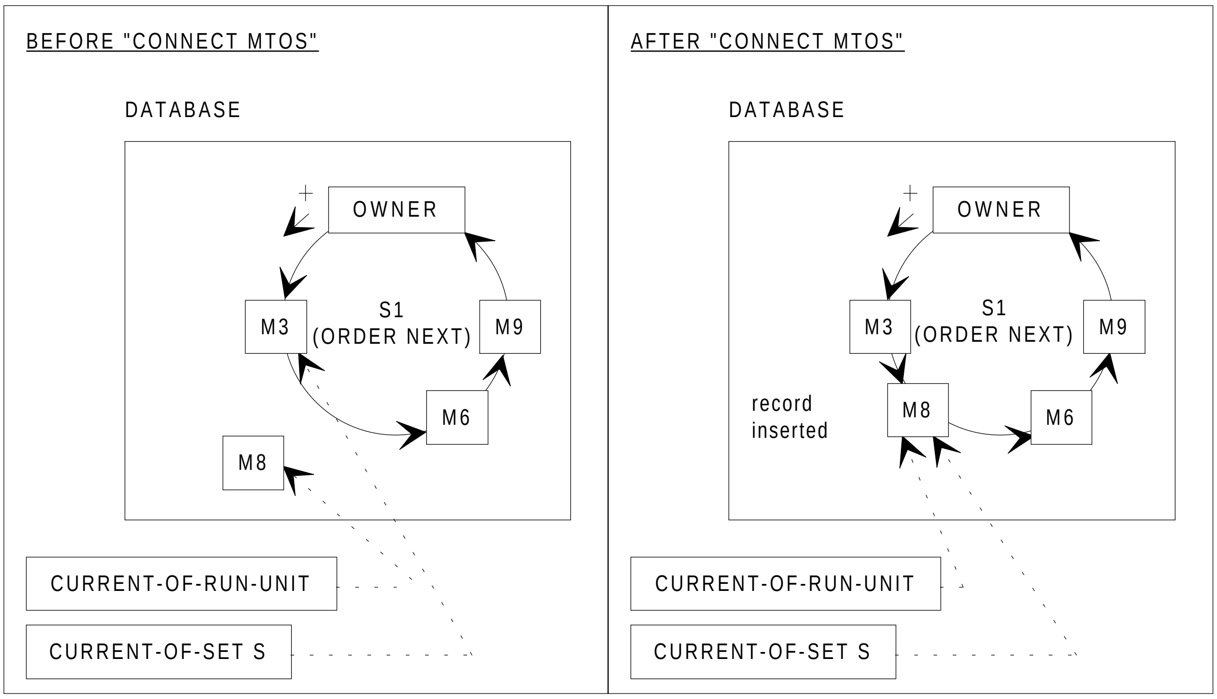
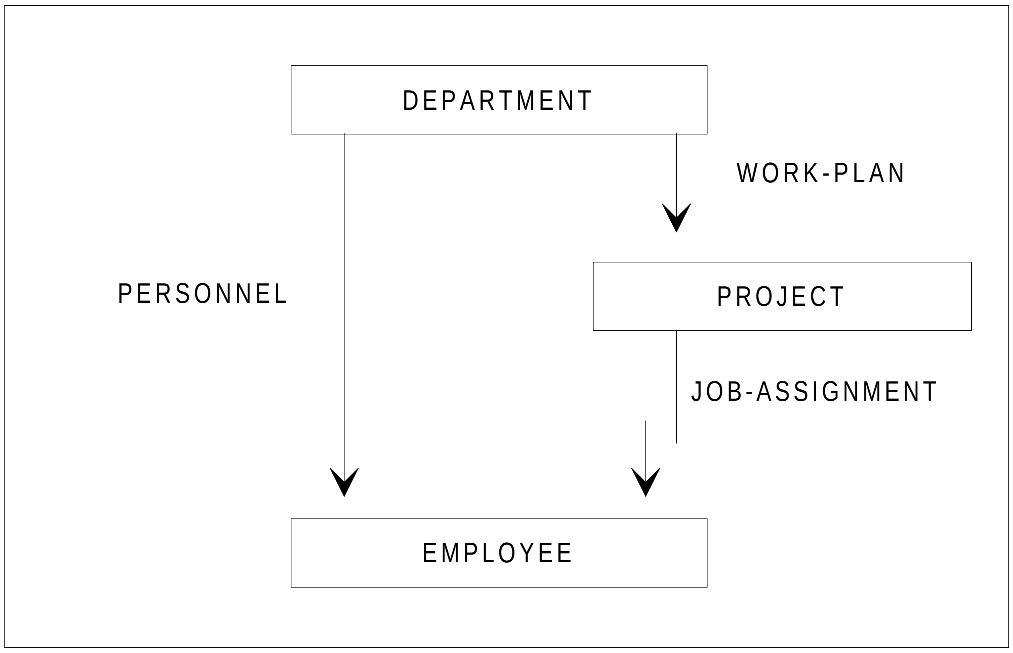


Figure 6-28. CONNECT Function

6.4.2 Example

Consider the following data structure:



and the case of newly-hired employee 5031 who entered the "Engineering" department three months ago and is now assigned to project X after his period of probation.

The Schema DDL clauses relevant to the explanation are:

```
RECORD NAME IS EMPLOYEE
    LOCATION MODE IS CALC USING E-NUMBER DUPLICATES NOT
    WITHIN AREA-2.
02 E-NUMBER TYPE IS SIGNED PACKED DEC 7.
    ...
RECORD NAME IS PROJECT
    LOCATION MODE IS CALC USING P-NAME DUP NOT
    WITHIN AREA-1.
02 P-NAME TYPE IS CHAR 20.
    ...
SET NAME IS JOB-ASSIGNMENT
    OWNER IS PROJECT
    ORDER IS PERMANENT INSERTION IS LAST.
MEMBER IS EMPLOYEE
    INSERTION IS MANUAL RETENTION IS OPTIONAL
    SET SELECTION IS THRU JOB-ASSIGMENT
    OWNER IDENTIFIED BY CALC-KEY.
```

The DML sequence reflecting the assignment of employee 5031 to project X might be the following:

```
MOVE 5031 TO E-NUMBER.             Establish current
FIND ANY EMPLOYEE.                 of run-unit.
MOVE "X" TO P-NAME.                Prepare set selection
                                   path.
CONNECT EMPLOYEE TO JOB-ASSIGNMENT. Connect to project X.
```

6.4.3 When the CONNECT Function is Impossible

All the SET SELECTION clauses activated by the CONNECT statement must be acceptable for processing by the DBCS. Any clause which is not acceptable will be detected either at compile-time if the record-type is specified, as in the example below:

```
CONNECT R01.
```

or at run-time if the record-type is not specified, as shown in the following example:

```
CONNECT.
```

If this is the case, the CONNECT statement is denied and a data-base-exception occurs with the following status code:

```
DB-STATUS : 0204200
               ( THE SUB SCHEMA DOES NOT INCLUDE A DATA ITEM
               OR SET-TYPE REQUIRED BY SET SELECTION OR SET
               ORDERING CRITERIA )
```

6.4.4 The CONNECT Function with a Detachable Index

If a SET SELECTION clause which is activated by the CONNECT statement invokes a secondary key declared in a detachable index, the DBCS performs a consistency check on the index which compares the data-base-key of the selected record, found via the index retrieval path, to the database record. If the data-base-key which is found leads to a missing record in the database, or to a different record type, the CONNECT statement is denied and a data-base-exception occurs with the following status code:

```
    DB-STATUS :  0273615       ( INDEX INCONSISTENCY )
```

6.5 DISCONNECT

6.5.1 General

The DISCONNECT function **removes a record from a set in which it has been declared as an OPTIONAL member.**

Its format is:

```
    DISCONNECT [record-name] FROM set-name
```

The object record of the statement is the current-of-run-unit. If **record-name** is specified, the DBCS checks that the current-of-run-unit is of this record-type.

The object record of the DISCONNECT statement must be declared in the schema as an OPTIONAL member of the set-type specified by **set-name**. It must be currently connected to an occurrence of this set-type if it is a MANUAL member.

A DISCONNECT statement which has executed successfully has the following three effects:

1) If the record disconnected was the current of the set-type specified by **set-name**, the corresponding current-of-set-type is updated to identify the relative position within the set of the disconnected record. This relative position is a **virtual pointer**, since it does not point to the record, but memorizes the data-base-keys of the NEXT, PRIOR and OWNER records of the disconnected record. This enables the program to use this virtual current-of-set-type in subsequent DML statements such as:

```
FIND NEXT (PRIOR,OWNER,FIRST,LAST,integer...) WITHIN set-name
```

The "FIND CURRENT WITHIN set-name" option of the "FIND... WITHIN set-name" statement is prohibited, however, since it attempts to retrieve the disconnected record.

Figure 6-29 illustrates this concept of virtual pointer.

2) If the record disconnected was not the current of the set-type specified by **set-name**, the corresponding current-of-set-type is not affected.

3) All other currency indicators are left unchanged.

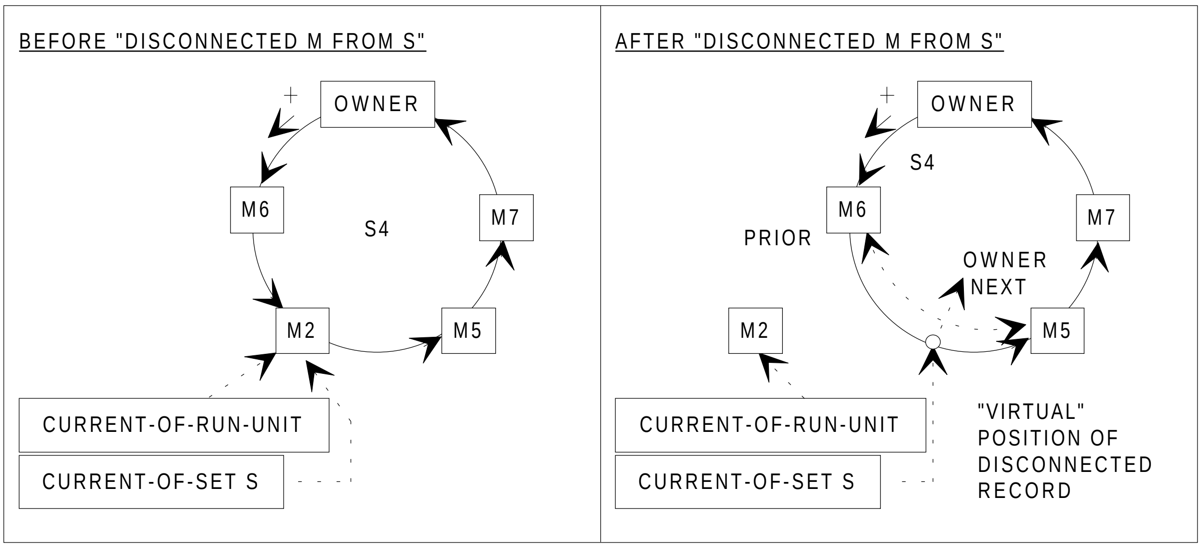


Figure 6-29. Virtual Current-of-set-type after DISCONNECT Statement

6.5.2 Example 1

Suppose that all the OPTIONAL members of a given set occurrence are to be disconnected (see Figure 6-30).

Let us assume that the "current-of-set S" is initially the owner A1 of the set occurrence S4 to be emptied. The disconnection sequence could be:

```
LOOP.    FIND NEXT B WITHIN S.
    IF DB-STATUS = "0502100" GO TO END-LOOP.
    DISCONNECT B FROM S.
    GO TO LOOP.
END-LOOP.    ...
```

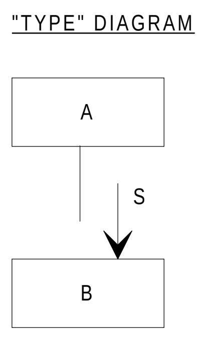
The currency chart in Figure 6-30 shows the evolution of the currency indicators during the execution of the DISCONNECT statement and illustrates the use of a virtual current-of-set-type. Further explanations of DML statements listed in the currency chart are given below.

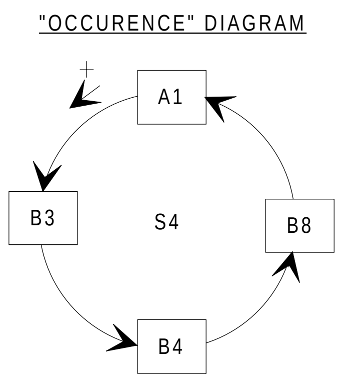
The first **FIND NEXT B WITHIN S** statement starts from the owner A1 and retrieves B3 which becomes current-of-run-unit and the "current-of-set S".

DISCONNECT B FROM S removes B3 from S4. As B3 was the current record of set S, the "current-of-set S" becomes virtual and functionally points somewhere between A1 and B4.

FIND NEXT B WITHIN S starts from the virtual "current-of-set S" and retrieves B4.

The processing goes on until an end-of-set condition occurs.





"CURRENCY" CHART

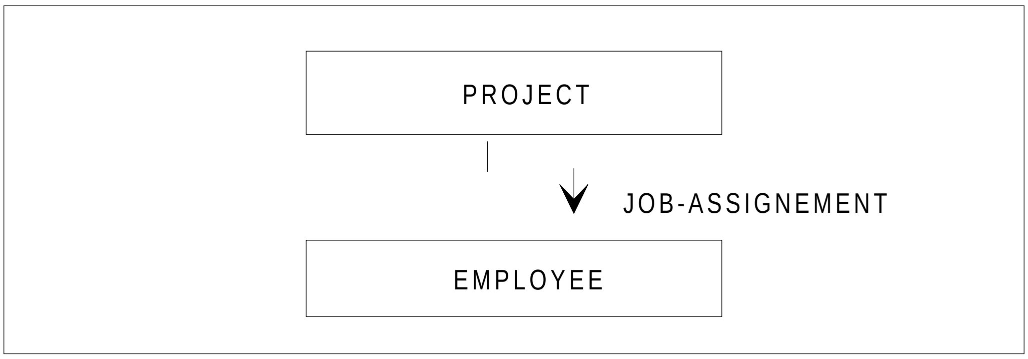
Symbol "-" means that the currency is not altered by the DML statement

DML STATEMENT	CURRENT OF RUN-UNIT	CURRENT OF-SET S
initial conditions	–	A1
FIND NEXT B WITHIN S	B3	B3
DISCONNECT B FROM S	–	Virtual
FIND NEXT B WITHIN S	B4	B4
DISCONNECT B FROM S	–	Virtual
FIND NEXT B WITHIN S	B8	B8
DISCONNECT B FROM S	–	Virtual
FIND NEXT B WITHIN S (EOS)	–	-

Figure 6-30. DISCONNECT Sequence

6.5.3 Example 2:

Consider the following data structure:



The problem to solve in this example is to transfer employee 727 from project X to project Y (see Figure 6-31, on the following page).

We assume the following about records PROJECT and EMPLOYEE:

- that they are located in only one area
- that their location mode is CALC with DUPLICATES NOT
- that their CALC-key is PROJECT-NAME and EMPLOYEE-NUMBER respectively
- that the set selection path has only one level defined BY APPLICATION.

The sequence used to perform the transfer might be:

```
MOVE "Y" TO PROJECT-NAME.                           Locate project Y as
FIND ANY PROJECT.                                   "current-of-set
MOVE 727 TO EMLOYEE-NUMBER.                          JOB-ASSIGNMENT"
FIND ANY EMPLOYEE RETAINING CURRENCY FOR SETS.      Locate employee 727
DISCONNECT EMPLOYEE FROM JOB-ASSIGNMENT.            as "current-of-run-unit"
CONNECT EMPLOYEE TO JOB-ASSIGNMENT.                 Change set occurrences
```

FIND ANY PROJECT establishes project Y as "current-of-set JOB-ASSIGNMENT" for the subsequent CONNECT statement.

FIND ANY EMPLOYEE RETAINING CURRENCY FOR SETS establishes employee 727 as current-of-run-unit but maintains project Y as "current-of-set JOB-ASSIGNMENT".

DISCONNECT EMPLOYEE FROM JOB-ASSIGNMENT does not affect any currency indicator, since employee 727 was not the "current-of-set JOB-ASSIGNMENT".

CONNECT EMPLOYEE TO JOB-ASSIGNMENT connects employee 727 to the set occurrence J5, which is determined by its current-of-set-type (in other words, by project Y). After insertion according to the set ordering criteria, employee 727 becomes the "current-of-set JOB-ASSIGNMENT".

The problem could have been solved using the following MODIFY statement instead of using DISCONNECT and CONNECT statements:

MODIFY EMPLOYEE ONLY JOB-ASSIGNMENT MEMBERSHIP.

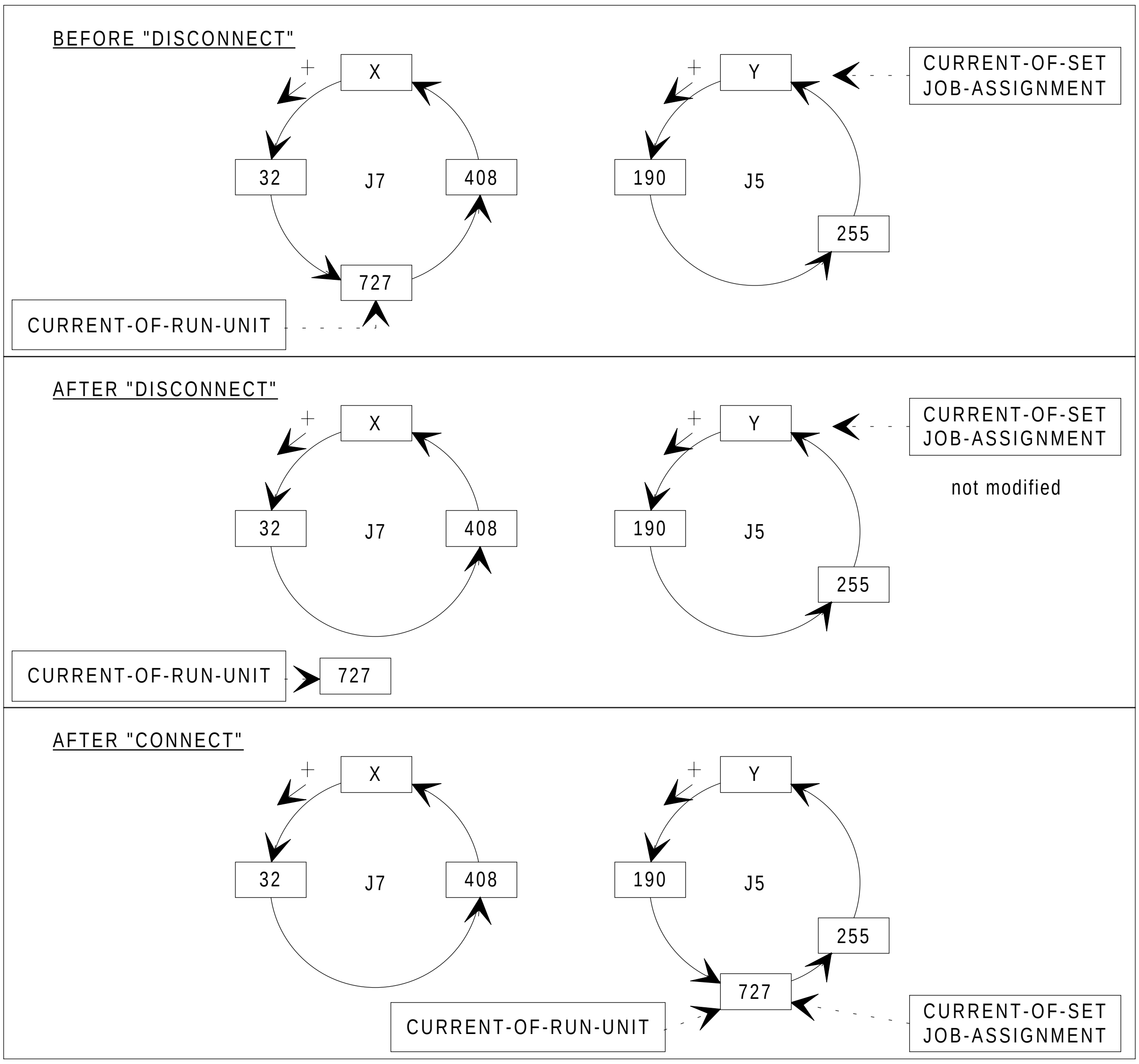


Figure 6-31. Changing Set Occurrences with DISCONNECT and CONNECT

7. Data Manipulation Control Functions

This section describes the three data manipulation control functions: READY, FINISH and USE.

The **READY function** makes one or more areas available for processing. It specifies the mode in which these areas can be shared with concurrent run-units and whether the areas will be objects of either retrieval or update operations.

The **FINISH function** terminates the processing of one or more areas.

The **USE function** centralizes the processing of data-base-exceptions.

These functions are described in detail in the body of this section.

7.1 READY

7.1.1 General

The READY function **makes one or more areas available for processing**. For database files, the DML READY statement is the equivalent of the COBOL OPEN statement for other files.

The READY function also makes associated index areas (index areas which implement a secondary key defined on a readied area) available for processing by the DBCS.

Its format is:

```
      [                                     {EXCLUSIVE {RETRIEVAL}} ]
      [                                     {          {         }} ]
      [                                     {          {UPDATE   }} ]
      [                                                             ]
READY [ [realm-name-1] . . . USAGE-MODE IS {SHARED      RETRIEVAL } ]
      [                                                             ]
      [                                    {           {RETRIEVAL}} ]
      [                                    {           {         }} ]
      [                                    {[MONITORED]{UPDATE   }} ]
```

If **realm-name** is coded, only the areas specified are made available. A realm-name can only appear only once if several usage modes are specified in the READY statement. If **realm-name** is omitted, all the areas of the schema are made available.

The USAGE-MODE Clause specifies the sharing mode and the access mode of the corresponding areas. The **sharing mode** is the mechanism which specifies whether concurrent run-units are allowed to access the areas. The sharing modes are listed and explained below:

- EXCLUSIVE mode - In this mode, the run-unit requires exclusive control of areas.
- SHARED mode - In this mode, the run-unit accepts concurrent run-units which perform retrieval operations, however the run-unit can **only** perform retrieval operations.
- MONITORED mode - In this mode, the run-unit accepts concurrent run-units performing retrieval **or** update operations. Further details are given in Section VIII of this manual.

  The **access mode** is the mechanism which specifies whether the run-unit will perform retrieval or update operations on areas. There are two access modes: RETRIEVAL and UPDATE.

- RETRIEVAL mode - This mode allows the following functions: FIND, GET, ACCEPT and DB Condition (IF)
- UPDATE mode - This mode allows all of the above functions, and the following functions in addition: STORE, MODIFY, ERASE, CONNECT and DISCONNECT.

Following the successful completion of a READY statement, the affected areas are in the READY state.

The sharing and access modes of corresponding index areas are set to values which allow the maximum amount of functions, depending on the sharing and access modes of related areas. For example, if an index is associated with two areas:

- one whose access mode is RETRIEVAL
- and another whose access mode is UPDATE

The index will have an UPDATE access mode set by the DBCS.

7.1.2 Sequencing of DML Statements

The counterpart of the READY statement is the FINISH statement, which is analogous to the COBOL CLOSE statement. A FINISH statement, therefore, places an area in the state in which it is no longer available for processing. An area is in the FINISH state at step initiation or after a successful FINISH statement which referenced it.

The opening (READYing) of the database is triggered either by the execution of a READY verb or by a secondary program, whichever comes first. The closing (FINISHing) of the database is triggered by the execution of a FINISH verb, applied to the last area currently in the READY state.

The following rules must be respected in the sequencing of DML statements.

1) When the database is closed, the only DML statement allowed is the READY statement, otherwise a data-base-exception occurs.

2) The execution of a READY statement fails if an area implicitly or explicitly referenced is already in the READY state. If the usage mode of an area is to be changed, the area must first be closed before being readied again in the new mode.

3) The execution of a statement other than READY fails if the areas involved in the processing of the statement are not in the READY state or not in the access mode required. When selecting the areas to be readied in a program, one must be aware that:

   a) The DBCS may follow a set selection path which involves other areas than the area containing the eventual set owner. Such areas must be readied at least in RETRIEVAL mode.
   b) The DBCS may update set pointers in areas other than the area containing the object record of an update function. Such areas must be readied in UPDATE mode.

7.1.3 An IDS Session

The period during which the database is open is called an IDS session.

Several IDS sessions which reference the same database, in one or several programs, may follow in sequence during a step.

Example:

```
                  { READY USAGE-MODE IS EXCLUSIVE RETRIEVAL.
1st IDS session   { . . .
                  {FINISH .

                  { READY A05 USAGE-MODE IS EXCLUSIVE UPDATE.
2nd IDS session   { . . .
                  {FINISH A05 .

                  {READY A05 USAGE-MODE IS EXCLUSIVE.
RETRIEVAL .       {
  3rd IDS session { READY A08 USAGE-MODE IS EXCLUSIVE UPDATE.
                  { . . .
                  { FINISH A05 A08.
```

Several IDS sessions referencing different databases using different programs, may be disjoint or nested or may overlap during the step.

Statistical information is collected for each IDS session.

7.1.4 First READY Statement of an IDS Session

The first READY statement of an IDS session has a particular role if the data base is not yet open : it incorporates the initialization of the session with the opening of areas. During this initialization phase, the DBCS performs the following operations:

- it reads the object schema file and loads the run-time control structures.
- it reads the IDSOPT input enclosure or file, if any, for the run-time options.
- it optionally opens the IDSTRACE file.
- it initializes all the currency indicators to a NULL value.

7.2 FINISH

7.2.1 General

The FINISH function terminates the processing of one or more areas. For database files, the DML FINISH statement is the equivalent of the COBOL CLOSE statement for other files.

Its format is:

```
    FINISH [realm-name]
```

If realm-name is coded, only the areas specified are put into a state in which they are unavailable for processing. If realm-name is omitted, all the areas of the schema are made unavailable for processing.

The execution of a FINISH statement fails if an area implicitly or explicitly referenced is not in the READY state.

Following the successful completion of a FINISH statement, the currency indicators which had been pointing to records within the finished areas are set to NULL.

7.2.2 Last FINISH Statement of an IDS Session

The last FINISH statement of an IDS session plays a particular role: it incorporates the termination of the session with the closing of areas. During this termination phase, the DBCS performs the following operations:

- it unloads the run-time control structures
- it optionally closes the IDSTRACE file
- it optionally displays the statistical and timing information related to the IDS session in the JOR

7.3 USE

7.3.1 General

The USE function **centralizes the processing of data-base-exceptions**. For database files, the DML USE statement is the equivalent of the COBOL USE AFTER ERROR statement for other files.

Its format is:

```
USE FOR DB- EXCEPTION
```

The DML USE statement is optional. If it is specified, it must appear only once and **in the first section** of the DECLARATIVES part of the PROCEDURE DIVISION. The syntax rules are the same as for any other COBOL USE statement.

When a data-base-exception occurs (whenever DB-STATUS NOT = "0000000") the DBCS transfers control to the section corresponding to the USE statement, if any. The procedures within this section can analyse the error and decide whether it is non-fatal (end-of-set, end-of-realm, record-not-found) or fatal.

If these procedures transfer control back to the non-declaratives part, they will have set a status flag specific to the program, in which each value represents one or more data-base-exceptions.

If they do not transfer control back to the non-declaratives part, they can display the information contained in the database special registers or other useful data. These procedures can also set an abnormal completion code (call to H_CBL_USETST) or a switch for the following JCL steps, and then stop the program.

7.3.2 Example

Suppose that, in a retrieval program, only the "end-of-set", "end-of-realm", "set selection failed" and "record selection failed" conditions are reported to the caller, using a specific status flag. The COBOL statements to handle the data-base-exceptions might be written as indicated in Figure 7-1.

```
DATA DIVISION.
            ...
   77 STATUS-FLAG PIC 9 VALUE 0.
        88 END-OF-PATH VALUE 1.
        88 REC-NOT-FOUND VALUE 2.
            ...
PROCEDURE DIVISION.
DECLARATIVES.
DB-ERROR-PROCESSING SECTION.
             USE FOR DB-EXCEPTION.

DB-TEST.
          IF DB-STATUS = "0502100" GO TO SET-END-OF-PATH.
            (end of set or realm)

          IF DB-STATUS = "0502300" GO TO SET-REC-NOT-FOUND.
            (set selection failed)

          IF DB-STATUS = "0502400" GO TO SET-REC-NOT-FOUND.
            (record selection failed)

DB-ABORD.
          DISPLAY ... UPON SYSOUT. (display useful information)
          SET SWITCH-1 TO ON. (set switch for JCL flow)
          STOP RUN. (abort program)

SET-END-OF-PATH.
          MOVE 1 TO STATUS-FLAG.
          GO TO DB-TEST-EXIT.
SET-REC-NOT-FOUND.
          MOVE 2 TO STATUS-FLAG.

DB-TEST-EXIT. EXIT. (return after DML statement)
END DECLARATIVES.
DB-RETRIEVAL SECTION.
             ...
          FIND FIRST WITHIN AREA-2.
          IF END-OF-PATH GO TO AREA-EMPTY.
             ...
AREA-EMPTY.
          MOVE 0 TO STATUS-FLAG. (reset flag for subsequent tests)
             ...
          FIND R6 WITHIN SI0 USING R6-NAME.
          IF REC-NOT-FOUND GO TO RECORD-ABSENT.
             ...
RECORD-ABSENT.
          MOVE 0 TO STATUS-FLAG.
             ...
          END COBOL.
```

Figure 7-1. Handling of Data-base-exceptions

8. Complementary Notions on Data Manipulation

This section describes other aspects of data manipulation which are not covered explicitly by the CODASYL proposals.

The first part presents the possibilities of concurrent access to the same database by several run-units.

The second part deals with the intercommunication between several programs accessing the same database within the same run-unit.

The third part presents the possibilities of access to several databases by the same run-unit.

The fourth part describes the global consistency checks which are performed at run-time to ensure that the necessary IDS/II components are brought together for execution. It also lists the conditions in which some of these checks may be disabled.

8.1 CONCURRENT ACCESS TO THE SAME DATABASE BY SEVERAL RUN-UNITS

8.1.1 Purpose

The need for concurrent access to a database comes from the integration of data and from the trend towards a transactional mode of operation.

8.1.1.1 Data Integration

Applications involving conventional files used to have their own data and could run concurrently in multiprogramming mode without actually sharing their data. New applications based on the database approach are now compelled to access the same common data when they run concurrently.

8.1.1.2 Transactional Mode of Operation

In a batch environment, there are few concurrent jobs in the machine. Concurrent access can be avoided without reducing response time by serializing the jobs which are accessing the database.

In a transactional environment, there may be several dozen users and response time is critical. It is therefore absolutely necessary to coordinate the activities of all these users.

8.1.2 Sharing Requirements

In the most general case of database operation, that is, when update is allowed, certain basic requirements must be met to operate a system of concurrent access to a database. First, the design of a program must account for the possible interaction of other programs in terms of data consistency and response time. Secondly, the concurrent access mechanism entails overhead time during execution.

Nevertheless, there are specific modes of operation which imply restrictions on sharing and in which the programming constraints and system overhead can be avoided.

To cover these different cases, three sharing modes are required:

- an exclusive mode
- a mode allowing n readers
- and a mode allowing n readers and p writers

In the following presentation of these sharing modes, a **user** is a competitor for access to the database, a **run-unit** designates the execution of one or more programs on behalf of a user.

8.1.2.1 Exclusive Sharing Mode (see the upper portion of Figure 8-1)

In this mode, no sharing is allowed: the user has exclusive control of the database. The exclusive mode, when employed by a run-unit for update operations, has the following advantages:

- The protection of the run-unit against others: the only modifications to the database are those which it performs.

- The protection of the other run-units: the database is made available to other run-units only when it is in a consistent state, after the completion of all the modifications.

For retrieval operations, the exclusive mode is not necessary, since it only permits a serialization of the run-units. It is more advantageous to use the **n readers** mode for such operations.

For efficient database operations, the exclusive mode is particulary adapted to mass update; it is the required mode for global restoration (FILREST, FILDUPLI).

8.1.2.2 "n Readers" Sharing Mode (see the center portion of Figure 8-1)

In **n readers** mode, each user performs only retrieval operations. This mode assumes that no modifications are being perfromed by other run-units. Many concurrent readers causes neither programming constraints nor system overhead.

For efficient database operations, the **n readers** mode is adapted to mass retrieval, global validation (VALIDATE command of DBUTILITY) or global save (FILSAVE, FILDUPLI).

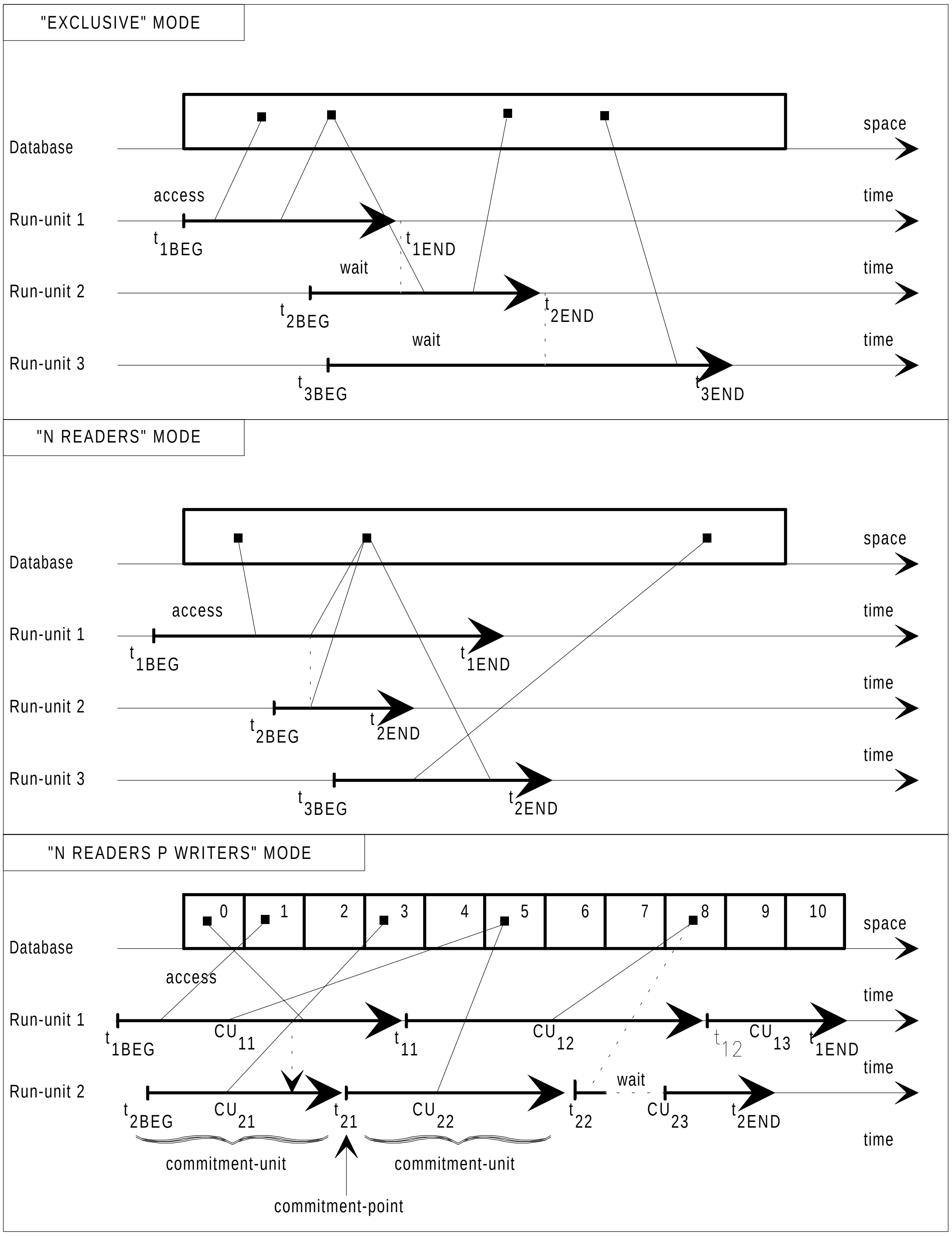


Figure 8-1. SHARING MODES

8.1.2.3 "n Readers p Writers" Sharing Mode (see the lower portion of Figure 8-1)

In this mode, each user is either a reader or a writer, but the logic of his programs assumes that modifications may be performed by other run-units.

In the **n readers p writers** mode, all concurrent users do not have access to the **entire** database simultaneously, although this is the user's impression. This mode operates by breaking down the run-units and the database itself into smaller independent parts. Each run-unit is considered as a sequence of phases called commitment-units which are separated by commitment-points. The database is considered as a group of pages. A run-unit accessing the database is thus transformed into the succession of commitment-units accessing pages.

If concurrent commitment-units are short and if they access pages randomly throughout the database, there is a high probability that they will work on separate parts of the database, thus operating on separate files and causing no mutual interference. Nevertheless, conflicts are possible, which require the presence of a page reservation mechanism capable of automatically handling delay and deadlock problems.

This mechanism reserves pages when required during the commitment-unit and releases them at the commitment-point. When a commitment-unit is denied access to a page which is held by another commitment-unit, it is put in the holding state until the first commitment unit terminates. If a deadlock situation occurs:

- the active commitment-unit is aborted
- the run-unit and the affected part of the database are returned to the state in which they were found at the last commitment-point
- and the pages reserved up to the time of the abort are released.

The run-unit is then restarted automatically.

Within a commitment-unit, the run-unit can work in exclusive or **n readers** mode on the part of the database which it accesses. Pages modified are always held in exclusive mode. This ensures the data consistency required by the logic of the commitment-unit. This is the same logic of a single IDS verb (for example, for the consistent update of set pointers) or that of the user program (for example, for the consistent update of balanced accounts).

The logic of the run-unit as a whole must take into account the fact that commitment-unit n+1 cannot rely on the modifications performed by commitment-unit n, because there is the constant possibility of subsequent modifications being made by concurrent run-units.

From an operational point of view, the **n readers p writers** mode is adapted to the transactional mode of operation. It can be extended to a local validation or restoration (VALIDATE PAGE or PATCH command of DBUTILITY) while the database is on-line. Batch/IOF run-units accessing the database during a TDS session must respect the restrictions caused by the commitment-unit mechanism.

8.1.3 The Implementation of Sharing Modes

The following subsection discuss the sharing modes in relationship to the operation and implementation of GCOS 7.

8.1.3.1 User/Run-unit/Commitment-unit

In Batch environment, the user is the person who runs a batch job. In IOF environment, the user is the person performing the IOF session. The batch job or IOF session is composed of one or more steps; each step constitutes a run-unit. Within the run-unit, it is possible to define commitment-points by programming calls to the system function wich is called H_GAC_UCOMIT. If no such call is coded, the step is considered as a single commitment-unit. In all cases, the end of a step is a commitment-point.

In a TDS environment, the concepts are less easily isolated. The TDS step itself runs on behalf of the user who launched it and is a run-unit on the same basis as other steps. However, TDS is a subsystem which permits concurrency between all the users connected to it. The terminal session of such a user constitutes a run-unit in its own right. This run-unit is a sequence of transactions split into commitment-units. The definition of commitment-points is done at TDS generation time or at run-time through calls to DFCMIT.

To summarize, in a Batch/IOF step there is only one commitment-unit at a given time whereas in a TDS step there are several commitment-units running in parallel.

8.1.3.2 Database Sharing Levels

A database is composed of several storage areas; these areas are UFAS integrated files. The file is the unit of sharing. Therefore a given run-unit A accessing areas M and N of the database and another run-unit B accessing areas P, Q and R of the same database are not considered as actually sharing the database; they can run concurrently without any interaction.

The sharing mode requested by a user is indicated at two levels: the file level and, when applicable, the page level. In the following discussion concerning files, general descriptive terminology is employed. Please refer to the **IDS/II Reference Manual** for specific terms such as DML, JCL, and TDS generation.

8.1.3.3 File level

At the file level, there are three sharing modes corresponding to those previously described:

- the **exclusive mode** is termed EXCLUSIVE
- the **n readers** mode is termed SHARED
- the **n readers p writers** mode is termed MONITORED

File Management (FIMA) is the mechanism which is responsible for controlling the access at the file level. This access control occurs at ASSIGN time.

Figure 8-2 shows the decision table for the acceptance of a new user.

area current state new user request	**inactive**	**held in EXCLUSIVE**	**held in SHARED**	**held in MONITORED**
EXCLUSIVE	Y	-	-	-
SHARED	Y	-	Y	-
MONITORED	Y	-	-	Y

Figure 8-2. Access Control at File Level

If the area is inactive, the first user is granted access and the sharing mode which he requests is imposed. (We asssume that the sharing mode is not a constraint imposed by the catalog). As soon as the area is held by a user, other users are accepted only if they request the same sharing mode as the current one (except in the case of EXCLUSIVE mode). Otherwise the users have to wait. When a user is granted access to an area, he remains a tenant of the area until DEASSIGN.

FIMA considers users as steps. With TDS, there is an ambiguity between the sharing mode of the TDS step itself and the sharing mode of connected users. Operationally, connected users always work in MONITORED mode between themselves. The TDS step as a whole can work in MONITORED or EXCLUSIVE mode with respect to other steps.

In general, the ASSIGNs are static. The step knows in advance all the areas which it will utilize. The user who initiates the step issues a request for access on each of these areas. If all the requests are accepted, the step is started. If one request is denied, all the requests are cancelled and the step is put in the holding state. This method avoids a deadlock caused by concurrent steps requesting the same files in a different order.

When ASSIGNs are dynamic, that is, executed during a step as the result of an operator intervention, the detection and resolution of a deadlock are the operator's responsibility.

8.1.3.4 Page level

Sharing mode at the page level applies only when the sharing mode at file level is MONITORED; that is, when access control is actually exercised at the page level and uses the concept of commitment-unit.

Three sharing modes are available at the page level:

- EXCLUSIVE mode, which is an exclusive mode in terms of update or retrieval
- SHARED mode, which is also a shared mode in terms of update or retrieval
- STATISTICAL mode, which is a degraded mode for retrieval. In this mode no page reservation takes place. A copy of the disk page is given to the user even if other users have reserved it in EXCLUSIVE or SHARED mode. The consistency of the page copy is not guaranteed.

The Generalized Access Control (GAC) mechanism is responsible for controlling the access at the page level. The control occurs each time a page is requested during a commitment-unit.

Figure 8-3 shows the decision table for the acceptance of a new user. This table is somewhat simplified since the STATISTICAL Mode is not necessarily separate from the EXCLUSIVE or SHARED Mode.

area current state new user request	**inactive**	**held in EXCLUSIVE**	**held in SHARED**	**STATISTICAL**
EXCLUSIVE	Y	-	-	Y
SHARED	Y	-	Y	Y
MONITORED	Y	Y (copy)	Y (copy)	Y

Figure 8-3. Access Control at Page Level

When a user is granted access to a page in EXCLUSIVE or SHARED mode, he remains a tenant of the page until the end of the commitment-unit. When a user is denied access to a page, he is put in the wait state or aborted, in case of a deadlock.

A deadlock cannot be avoided at the page level because the pages which are accessed by a commitment-unit are not known at the beginning of the commitment-unit. Note that a deadlock is usually caused by the interaction of two users, but the actual occurrence of a deadlock may require the intervention of three or more users. Moreover, the pages at stake in a deadlock situation may be in the same area or in different areas (or both) in IDS areas and non-IDS files.

In all environments, the sharing mode at page level for updated pages is forced to EXCLUSIVE.

In batch and IOF environments, the sharing mode at page level for retrieved pages is specified at ASSIGN time. The same applies to all the commitment-units of the step. If STATISTICAL is selected, no update operation can be performed on any page by the run-unit.

In the TDS environment, the sharing mode at page level for retrieved pages is defined at TDS generation time for each transaction. It is the same for all the commitment-units of the transaction but may vary from one transaction to another. If STATISTICAL is selected, no page of the area can be updated by the transaction.

8.1.4 Using the Monitored Sharing Mode

This subsection discusses specific aspects of the MONITORED sharing mode.

8.1.4.1 Data Consistency

The MONITORED mode places constraints on the design of database applications. One constraint is that the programs acting on the database must be defined in terms of short and consistent global operations, each operation being composed of a sequence of elementary COBOL/DML instructions.

These operations must be short in order to access only a small part of the database, thus permitting the interleaving of concurrent run-units. They must be consistent: they must access and reserve all the data required to guarantee coherent database operations such as reading and incrementing counters, balancing of two accounts, etc.

If a global operation is to be mapped onto a commitment-unit, the GAC mechanism ensures the logical unity of the operation by ensuring the following:

- that pages reserved are not released until the commitment-point
- that these pages, once reserved, cannot be modified by concurrent commitment-units
- if a deadlock occurs, that the operation is rolled back and restarted from the beginning

The commitment-unit mechanism is a means which the system provides for guaranteeing the logical indivisibility of an operation. It is up to the user to define the contents and logic of the operation.

In a transactional environment, design constraints are fairly obvious: a TDS transaction is itself a single operation or a sequence of a few operations. These constraints are less adapted to a batch program. If such a program is likely to run in MONITORED mode, it is recommended to design it as an assembly of the global operations defined for the transactional environment. This assembly may be a sequence of different operations or a loop on the same operation, the limits of each operation being fixed by a commitment-point.

8.1.4.2 Page Reservation by the DBCS

Theoretically, page reservation is not the application designer's responsibility. He should think in terms of DML instructions directed to records and sets, and let the DBCS convert his instructions into requests for pages.

Concurrent access rules are implicitly defined by the implementation at the page level and by the algorithms of the DBCS. Since records and sets are not separate entities but instead take their form from logical definitions, concurrent access rules follow the internal logic of the DBCS. Thus, when the application designer studies concurrent access conflicts or estimates how many page transfers of the commitment-unit are required, he must think in terms of **pages** when trying to anticipate how the DBCS will carry out processing. He should keep in mind the following:

- When the DBCS accesses a storage record, it reserves the page that contains it, thus preventing modification of that page by concurrent run-units of the other records of the page, even if these records have nothing to do with the logic of the commitment-unit in progress.

- When the DBCS retrieves a storage record, it reserves the page which contains it in the mode defined for the step or transaction (EXCLUSIVE, SHARED, STATISTICAL).

- When the DBCS modifies a storage record, it reserves the page which contains it in EXCLUSIVE mode. The DBCS may first access a page in SHARED mode and then switch to EXCLUSIVE mode when it discovers that the page is to be modified.

- During the execution of a STORE statement, the DBCS may access several pages before finding room for the new record (multi-page CALC chain, search for free space for a DIRECT or VIA record).

- Each time a record is to be connected to a set (STORE of an AUTOMATIC member, CONNECT, or MODIFY MEMBERSHIP), the DBCS searches all the levels of the set selection path in order to identify the set occurrence. Then, depending on the set order, it either identifies the insertion point immediately or after a partial search of the set occurrence. When no-duplicate-control-fields exist, the entire set occurrence is searched. As the NEXT and PRIOR records need to be modified, their pages are reserved in EXCLUSIVE mode, in addition to the page of the inserted record itself. This is one influence of the set implementation on the concurrent access rules, because since the set pointers are modified in the NEXT and PRIOR storage record, the data part of these records cannot be modified by concurrent run-units. This restriction does not extend to records other than the NEXT and PRIOR records.

- Each time a record is disconnected from a set (DISCONNECT, MODIFY MEMBERSHIP, ERASE), the DBCS accesses in exclusive mode the page of the record and those of its NEXT and PRIOR records.

- During the execution of a FIND ANY DUPLICATE statement, the DBCS may access several pages if the CALC-chain overflows.

- During the execution of a FIND FIRST rec-n WITHIN realm statement, all the pages preceding the page containing the desired storage record are accessed. Concurrent run-units cannot store records in these pages. This guarantees the notion of FIRST for the duration of the commitment-unit. The same type of processing applies to the case of FIND LAST, NEXT, PRIOR WITHIN realm.

- During the execution of a FIND WITHIN set statement, the set occurrence is identified either directly by the current-of-set or after the search of the set selection path. Then the DBCS searches the occurrence, member by member, along an access path which starts from the owner or from the current-of-set.
- Each time a DML statement performs a direct access via a secondary key (FIND, STORE, MODIFY, SET SELECTION mechanism), the DBCS accesses the ROOT page of the associated index and that page leads to a given leaf page depending on the index level numbers.

8.1.4.3 Response Time/Wait/Deadlock

In a transactional environment, the problem of response must be considered. Here we will consider how the page access affects response time.

If there are no conflicts for pages between concurrent run-units, the response time of a commitment-unit, or of part of a commitment-unit which is broken down into several TPR's separated by exchanges, depends on the number of page transfers it performs. You cannot expect a good response time if the commitment-unit is designed as a batch program and carries out hundreds of I/0 operations on the database. The application designer must take into account not only of the number of DML functions of his programs, but also the operations performed by the DBCS for each DML function, since a complex function may trigger dozens of page transfers.

Response time is longer when page conflicts occur. The effects of an actual page conflict can be reduced by retrieving pages in SHARED mode, especially when transactions follow the same path in the database to retrieve common information. The STATISTICAL mode can also be used in certain circumstances but only when the portion of the database involved is stable (with permanent records).

A page conflict which causes a WAIT places the commitment-unit which contains the requested page in an non-active state with regards to the database. This commitment-unit is temporarily idle during an exchange over the telecommunications lines. The idle time is determined to a large extent by the think-time of the terminal operator. Under TDS, it is possible to make the distinction between a **short wait**, during which the tenant of the resource is active in a TDS process, and a **long wait**, during which the tenant of the resource is idle. In the second case, a message of the type "WAIT FOR RESOURCE" is sent to the operator of the waiting commitment-unit. A means of avoiding a **long wait** is to design commitment-units containing no intermediate exchanges or to concentrate the database accesses in the last TPR of commitment-units.

When a deadlock occurs, the commitment-unit which has requested execution is aborted and restarted from the beginning. If the commitment-unit contains no intermediate exchanges, the deadlock is not noticed at the terminal. If, however, intermediate exhanges have already taken place, the operator must receive a warning so that he knows that he must put in another request for execution.

To avoid this problem, it is advisable to break such a commitment-unit into two parts whenever the logic will permit such an break. There would thus be two logical divisions: one portion would send questions to the operator and collect his answers in the TRANSACTION STORAGE **without any database access**; the other would be dedicated to the actual database accesses. Thus, in the case of a deadlock, the second commitment-unit can be restarted unnoticed, using the answers which were recorded in the TRANSACTION STORAGE by the first commitment-unit.

Another solution for reducing the probability of deadlock is to design commitment-units so that they always request the same pages in the same order. This leads to a potential wait rather than to a deadlock. In such situations, it is recommended to read pages in EXCLUSIVE mode to avoid a another type of deadlock, which can occur when two commitment-units hold the same page in SHARED mode and both switch into the EXCLUSIVE mode to update it.

8.1.4.4 Transfer of Information between Commitment-units of the same Run-unit

Within a given run-unit, commitment-unit **n+1** should not logically rely on the state of the database which corresponds to the end of commitment-unit **n** because of the possible interaction of concurrent run-units between the executions of these two commitment units.

During real database operations, there are circumstances in which this rule does not apply. For example, a data-base-key will identify the same record in both commitment-units if it is known that this record is permanent. Therefore, this data-base-key may be transferred across the commitment-point so that FIND DB-KEY can rapidly access commitment unit n+1 a second time.

Rather than using a working area to transmit the data-base-key, it would be better to use the currency indicators. This function depends on the environment, as explained below.

In a TDS environment, currency indicators are always nulled at commitment-points. In such circumstances, a data-base-key must be transferred through the TRANSACTION STORAGE between commitment-units of a transaction; or through the PRIVATE part of the TRANSACTION STORAGE between transactions of the same run-unit.

In a BATCH/IOF environment, currency indicators are also nulled by default at commitment-points. However there exists a DBCS run-time command, the SAVE CURRENCIES AT COMMITMENT command, which allows the user to keep the contents of currency indicators across commitment-points. However, this command does not prevent the corresponding pages from being released at commitment-points. These pages are reserved again in the next commitment-unit if currency indicators pointing to them are used in DML functions.

Besides data-base-keys, other things must be considered. For example, the contents of a record can be passed across a commitment-point to avoid another access to the database at the next commitment-unit. An error at this level can jeopardize the consistency of the commitment-units taken as a whole.

8.2 COMMUNICATION BETWEEN PROGRAMS ACCESSING THE SAME DATABASE WITHIN THE SAME RUN-UNIT

8.2.1 Purpose

In certain database applications, it may be desirable to specifically write COBOL/DML subroutines to provide access to the database. This provides the user with the following advantages:

1) the ability to centralize the procedures using DML statements which facilitates debugging

2) the ability to provide calling programs with functions which are more elaborate than simple DML commands.

If the application is made up of several load-modules, these subroutines are to be linked to each load-module.

When a load-module is executed, the subroutines are considered as an integral part of the **run-unit**, not run-units accessing the database concurrently with the calling program; the call mechanism ensures the sequential execution of the programs and subroutines.

This implies that the calling programs and called subroutines share the same database context, composed of the following:

- currency indicators
- the UWA zones (DB-PARAMETERS, DB-CXT, DB-REGISTERS, record-descriptions)
- the run-time control structures
- and the buffers

The code of the DBCS and UFAS routines is also shared.

8.2.2 Main and Secondary Programs

In standard COBOL, a means of sharing UWA data zones between programs is to pass their addresses as parameters of the CALL. In an IDS/II database, this leads to the interaction between a **main program** and **secondary programs** (see Figurse 8.4).

- A **main program** is the program for which the UWA zones of the database are generated in the WORKING-STORAGE SECTION. The main program may perform any DML function and may be responsible in particular for the READY and FINISH functions if it runs as a batch job.

- **secondary programs** are programs for which the UWA zones of the databasse are generated in the LINKAGE SECTION. The secondary programs may perform any DML function.

Any main or secondary program may contain READY and FINISH verbs. Under TDS, however, the READY and FINISH verbs are optional and are ignored at execution time.

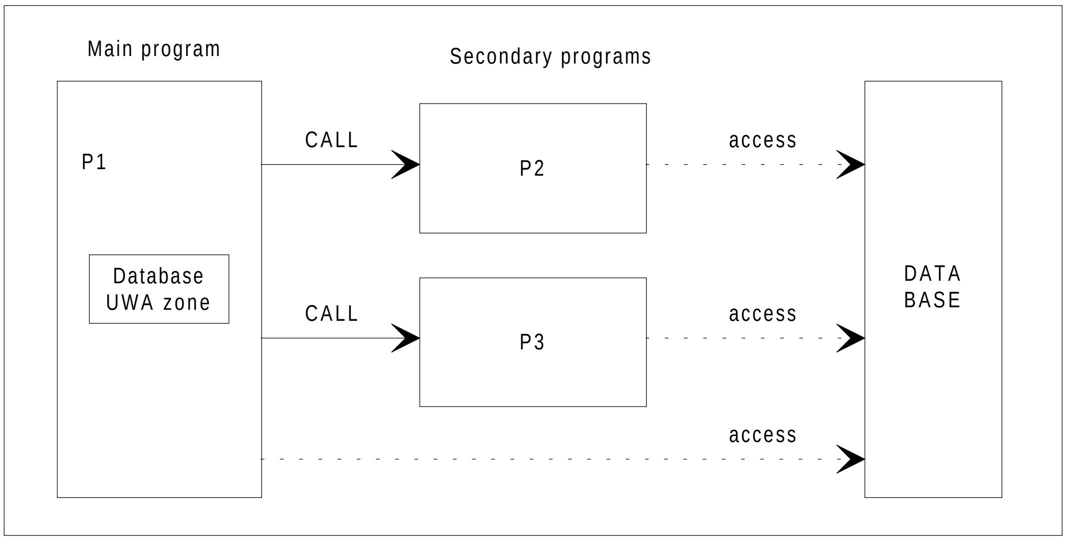


Figure 8-4. Main and Secondary Programs

The main program and the secondary programs can be viewed in an operating system context, when this IDS/II distinction of **main** and **secondary** no longer holds true. This is illustrated in Figure 8-5 and described below.

- First, the term **main** applies only to database environment. A main program can be called by another COBOL-only program which does not access the database.

- On the other hand, a secondary program can call another secondary program.

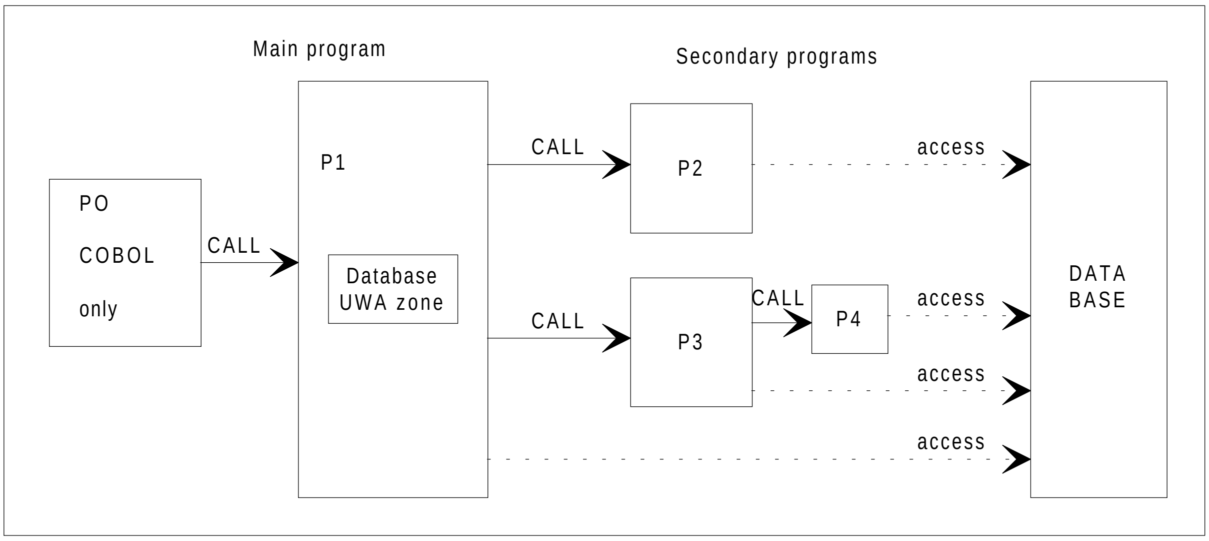


Figure 8-5. Generalization of the CALL Mechanism

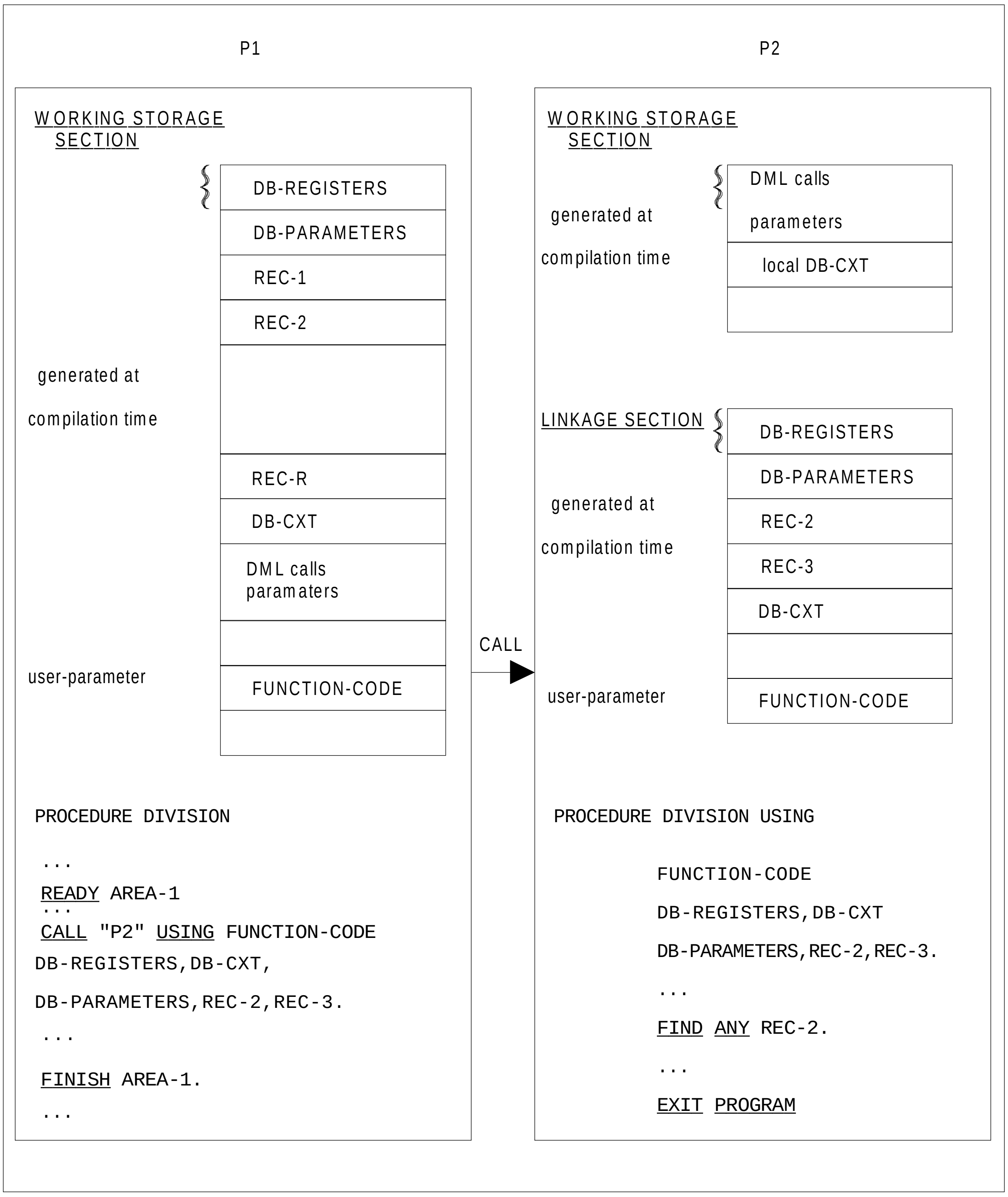
8.2.3 Programming Rules

8.2.3.1 Parameter Passing when a Secondary Program is Called

When a secondary program is called, the following parameters must be passed (see Figure 8-6):

- DB-REGISTERS
- DB-CXT (DBCS internal communication zone)
- DB-PARAMETERS, if there are any in the schema
- Record-descriptions. At least those used by the called program must be passeed. Note that there is no record-description for records whose user data part is void (this may occur for certain "root" or "relation" records).
- Any other user parameters.

All these parameters must be specified in the same order in the CALL statement of the calling program as in the PROCEDURE DIVISION Clause of the called program.



```
P1:
PROCEDURE DIVISION
 ...
 READY AREA-1
 ...
 CALL "P2" USING FUNCTION-CODE
DB-REGISTERS,DB-CXT,
DB-PARAMETERS,REC-2,REC-3.
 ...
 FINISH AREA-1.
 ...

P2:
PROCEDURE DIVISION USING
         FUNCTION-CODE
         DB-REGISTERS,DB-CXT
         DB-PARAMETERS,REC-2,REC-3.
         ...
         FIND ANY REC-2.
         ...
         EXIT PROGRAM
```

Figure 8-6. Parameter Passing

8.2.3.2 Main or Secondary Program Declaration

The DB-DESCRIPTIONS Clause of the SUBSCHEMA section directs the COBOL compiler in generating the UWA zones in the database and also defines the type of the program (see Figure 8-7):

If WORKING-STORAGE is specified, the program is a main program.

If LINKAGE is specified, the program is a secondary program.

If the DB-DESCRIPTIONS Clause is absent, a main program is assumed.

```
SUB-SCHEMA SECTION

{ DB db-ifn-name USING sub-schema-name
                 WITHIN schema-name.    } . . .

[                    {WORKING-STORAGE}                 ]
[ DB-DESCRIPTIONS IN {                }        SECTION .]
[                    {LINKAGE        }                 ]
```

Figure 8-7. DB-DESCRIPTIONS Clause

8.3 ACCESS TO SEVERAL DATABASES FROM THE SAME RUN-UNIT

8.3.1 Purpose

Certain situations may require that two or more databases be processed in the same run-unit. For instance, records may have to be retrieved from one database and stored into another one.

8.3.2 Implementation

In a given COBOL/DML program, it is possible to access more than one database. In this case, there are as many database contexts (UWA zones, currency indicators, run-time control structures) as databases. The buffers, however, can be pooled if the pool-names are the same. The code of the DBCS and Buffer Management routines is always shared.

Figure 8-8 gives an example of the DB clause of the SUB-SCHEMA SECTION necessary for the use of 2 databases:

1) schema-1 (via sub-schema-1)

2) and schema-2 (via sub-schema-2)

in the same COBOL/DML program:

```
SUB-SCHEMA SECTION


DB db-ifn-1     USING  sub-schema-1
                WITHIN schema-1.

DB db-ifn-2     USING  sub-schema-2
                WITHIN schema-2.
```

Figure 8-8. Access to Two Databases:Example

The db-ifn-name is only a **qualifier**.

If the same record name **rec-name** appears both in the sub-schema-1 and in the sub-schema-2 parameter, any statement which refers to rec-name is ambiguous. In order to remove this ambiguity, **rec-name** must be qualified with the appropriate db-ifn-name.

Example:

```
GET rec-name.             <-----  FORBIDDEN  (ambiguous)

GET rec-name of db-ifn-1. <-----  ALLOWED
```

NOTE: The use of a db-ifn is mandatory even if there is only one DB clause in the COBOL program. In this case, this db-ifn is a dummy literal.

8.4 GLOBAL CONSISTENCY CHECKS

8.4.1 Principles

The various components which participate in the IDS/II environment are prepared separately within the operating system context. First the Schema DDL and DMCL are translated to produce the object schema. Next, the Subschema SDDL is translated to produce the object subschemas. Then, the areas are preallocated using the object schema and the DML programs are compiled against one or several object subschemas.

In order to check at run-time that the right components are brought together, IDS/II places elements of reference in each component which characterizes the schema and subschemas. These elements of reference are described below.

8.4.1.1 Elements of Reference in the Object Schema

When translating the Schema DDL and DMCL source, DDPROC places three elements of reference in the object schema:

1) the schema name: by convention in a site, the schema name must uniquely identify a given database.

2) The DDL reference date-time: the date-time of the last successful DDL translation.

3) The DMCL reference date-time: the date-time of the last successful DMCL translation.

8.4.1.2 Elements of Reference in the Object Subschema

When translating the Subschema SDDL source, DDPROC places in the object subschema four elements of reference:

1) the schema name

2) the subschema name

3) the SDDL reference date-time: the date-time of the last successful SDDL translation.

4) the **validation date-time**: the date-time of the last successful SDDL validation (via the VALIDATE command of MNDD) or translation.

8.4.1.3 Elements of Reference in the Compiled Program

At compilation time, two elements of reference which are extracted from the object schema are placed in an internal communication zone (DB-CXT) of the working-storage by COBOL:

1) the subschema name

2) the SDDL reference date-time

8.4.1.4 Elements of Reference in an Area

When preallocating an area, two elements of reference which are extracted from the object schema are placed in the IDS label by PREALLOC:

1) the schema name

2) the DMCL reference date-time

8.4.1.5 Run-time Checks (see Figure 8-9 further on in this section)

Run-time checks verify the following:

1) the link between a program and the schema and subschema which are to be loaded for execution.

2) the link between an area and the schema loaded for execution.

When the first READY triggers the IDS session, the DBCS checks the following:

- That the subschema name and SDDL reference date-time of the main program match those of the loaded subschema. The date-time check is intended to ensure that the DML compilation time is later than the Subschema SDDL translation.

- That the loaded subschema validation date-time matches those of the loaded Schema DDL reference date-time. The date-time check is intended to ensure that the SDDL translation or validation time is later than the Schema DDL translation.

Each time an area is readied, the DBCS checks that the schema name and DMCL reference date-time of the IDS label match those of the loaded schema. The date-time check is intended to ensure that the area preallocation time is later than the Schema DMCL translation

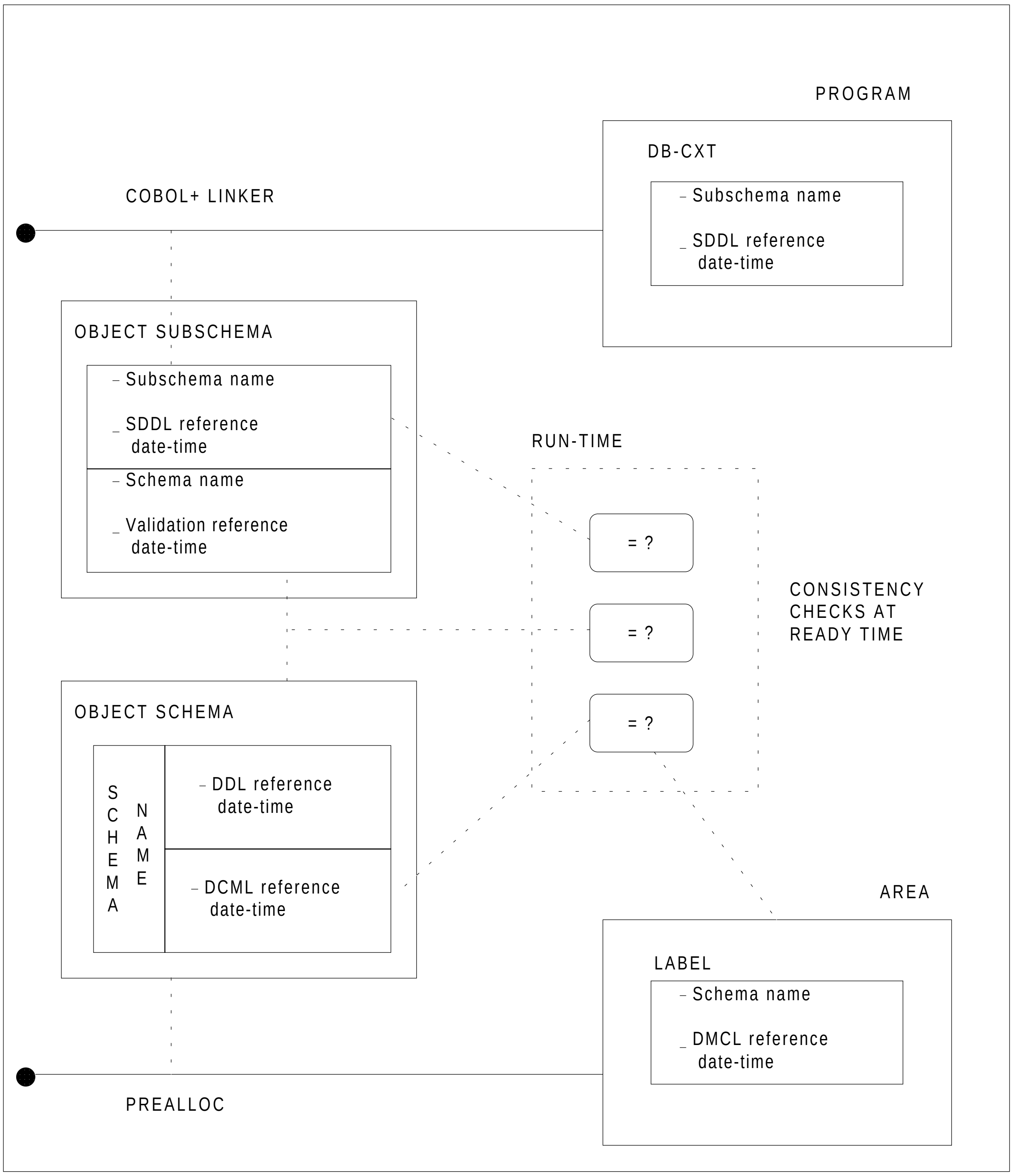

PROGRAM
DB-CXT
- Subschema name
_ SDDL reference date-time
COBOL+ LINKER
OBJECT SUBSCHEMA
- Subschema name
_ SDDL reference date-time
- Schema name
_ Validation reference date-time
RUN-TIME
= ?
= ?
= ?
CONSISTENCY CHECKS AT READY TIME
OBJECT SCHEMA
SCHEMA NAME
- DDL reference date-time
- DCML reference date-time
AREA
LABEL
- Schema name
_ DMCL reference date-time
PREALLOC

Figure 8-9. Consistency Checks at READY Time

8.4.2 Scope of the Consistency Checks Performed at READY Time

The COBOL compilation and area preallocation places information which is extracted from the object subschema into the generated programs and area labels. This information may remain in a symbolic form (record descriptions of DATA DIVISION) or may be converted into a more compact internal format (parameters of DBCS calls, area labels).

For example, the variable-length names appearing in the **SDDL** (areas, records, fields, sets, keys) are replaced by fixed-length code numbers. The **subschema code number** assignment is made in sequence, within each class of names, in the order of appearance of the SDDL entries/subentries that define these names.

In the **DDL**, the areas, records, fields, sets, keys, and db-parameters are also replaced by fixed-length code numbers. **The schema code number** assignment is made in sequence, within each class of names, in the order of appearance of the DDL entries/subentries that define these names.

In the **SDDL**, the subschema code numbers are associated to their corresponding schema code. All the subschema code numbers are used in the parameters of DBCS calls, and converted by the DBCS into schema codes.

The schema area code numbers are used in the area code part of the data-base-keys and IDS global pointers. The schema record code numbers are used in the header of storage records. The schema set code numbers are taken into account in the layout of the IDS pointer zone of storage records.

Thus, **the consistency checks performed at READY time are global checks** which guarantee that the IDS/II information recorded in internal format has the same meaning in the schema/subschema, the DML programs and in the database.

These checks guarantee **the consistency of IDS/II components at the structural level** (with regard to record-types and set-types), but not at the occurrence level. During database operations, the contents of the database areas evolve each time a DML program updates them. If for any reason, such as a selective restore operation, the different areas are not synchronized, inconsistencies will appear at run-time in the form of broken chains, record not found conditions, etc.

8.4.3 Means of Disrupting Global Consistency Checks

The checks described above assume the uniqueness of a schema and of a database occurence for a given application. Situations do arise, however, when schemas are not unique. Before discussing instances of such breaches of uniqueness, we will present the means which make the use of multiple schemas and database occurences possible, although certain performance are involved.

8.4.3.1 Imposing Elements of Reference onto the Object Schema

Given a reference schema, it is possible to define a new schema which is either identical to the reference schema or which differs from it in minor ways. In order to process a preexisting subschema with this new schema using preexisting DML programs or areas that contain the identification of the reference schema, one must:

1) give the new schema the same schema name.

2) stamp the new schema with the same DMCL reference date-time.

3) validate the preexisting subschema against the new schema.

The first condition is easily met when writing the DDL.

The date-time stamp is usually given to a schema when DDPROC reads the time from the computer timer at the moment when the DDL and DMCL are translated. This process ensures that these date-times cannot be reproduced inadvertently. To impose onto the new schema the same date-times as those in the reference schema, one must use the MNDD command: MODIFY_DATE.

The fact that date-time has been imposed is recorded in the new schema. Many checks are continuously carried out permanently by the DBCS routines. Nonetheless, inconsistencies may be uncovered after the opening time of the IDS SESSION, when the database has already been modified.

8.4.3.2 Imposing Elements of Reference onto the Object Subschema

Given a reference subschema, it is possible to define a new subschema which is either identical to the reference subschema or differing from it in some minor ways. In order to use preexisting DML programs which contain the identification of the reference subschema, one must:

1) give the new subschema the same subschema name

2) stamp the new subschema with the same SDDL reference date-time

3) validate the new subschema against the schema

The first condition is easily met when writing the SDDL.

To impose onto the new subschema the same date-times as in the reference subschema, one must use the MNDD command MODIFY_DATE.

The fact that date-time has been imposed is recorded in the new subschema.

8.4.3.3 Selection of Object Schemas and Areas with the ASSIGN Statement

Consistency checks are based on information residing in the object schema, DML programs and areas, but never on the external-file-names which identify their containers. Therefore, given a DML program, it is possible, using JCL, to choose the schema and the database against which it is to run. Figure 8-11 illustrates this for the schema but the case of the areas is identical.

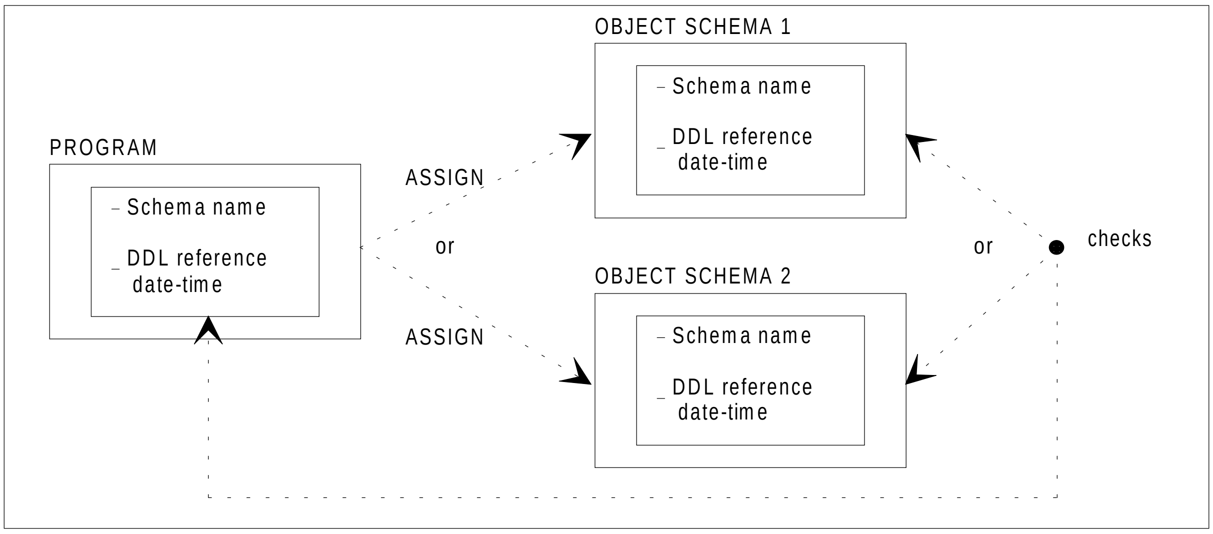


Figure 8-10. Checks Based on Contents, Not on Container Name

8.4.4 Situations which Require the Disruption of Global Consistency Checks

Let us now consider a few examples of situations in which it is necessary to make the global consistency checks ineffective.

8.4.4.1 Loss or Destruction of the Object Schema

If the object schema is destroyed by mistake, it is necessary to create a new schema identical to the one which has been destroyed. This is necessary so that existing DML programs can still be used without being recompiled, and so that the existing database can still be used without having to re-preallocate and reload.

The DDL must be the same as that of the former schema, including the order of the entries/subentries that define area, record, field and set names. The order of the db-parameters must also be the same. The DMCL must also be the same.

The DDL and DMCL values from the former schema for the reference date-times must be imposed on the new schema.

8.4.4.2 Loss or Destruction of the Object Subschema

If the object subschema is destroyed by mistake, it is necessary to create a new subschema identical to the one which has been destroyed, so that the existing DML programs can still be used without having to recompile.

The SDDL must be the same as that of the former subschema, including the order of the entries/subentries that define area, record, field, set names and keys.

The SDDL values from the former subschema for the reference date-times must be imposed onto the new subschema, and the subschema must be validated against the schema.

8.4.4.3 Database Application Check-out Phase

DML programs should be checked against an experimental database which is identical to or smaller than the final database.

8.4.4.4 If the Experimental Database is Identical to the Final Database

If the experimental database which is used for the check-out phase has exactly the same DMCL characteristics as the final database, the reference schema may be used for both databases. The areas of each database are preallocated against this schema. The selection of the database desired is done at run-time by JCL ASSIGN statements. Note that the consistency checks remain effective in this case.

8.4.4.5 If the Experimental Database is Smaller than the Final Database

If the experimental database which is used for the check-out phase is smaller than the final database, a check-out schema must be defined.

In the check-out schema, the DDL must be the same as that of the reference schema; the DMCL may differ by such parameters as NUMBER-OF-PAGES, PAGE-SIZE, LINES-PER-PAGE, record ranges, etc.

If the DDL part of the check-out object schema is obtained by a copy of the reference object schema, the DDL reference date-times will automatically be identical. If it is obtained by a retranslation of the DDL source, the values of the DDL reference date-time from the reference schema must be imposed on the check-out schema.

The DMCL reference date-time of the check-out schema may be different from that of the reference schema.

The areas of the check-out database must be preallocated against the new schema.

The DML programs are compiled against the reference schema. The program logic in these programs is not normally affected by changes in DMCL, however, if there are computations of data-base-keys among these changes, it is advisable to parameterize them using the ACCEPT statement that reads DMCL information from the schema.

8.4.4.6 Initial Loading of the Database

Considerations of improved performance may prompt the database administrator to design a schema which is better adapted than the schema of reference for the initial loading of the database, which contains the permanent records (See Figure 8-11).

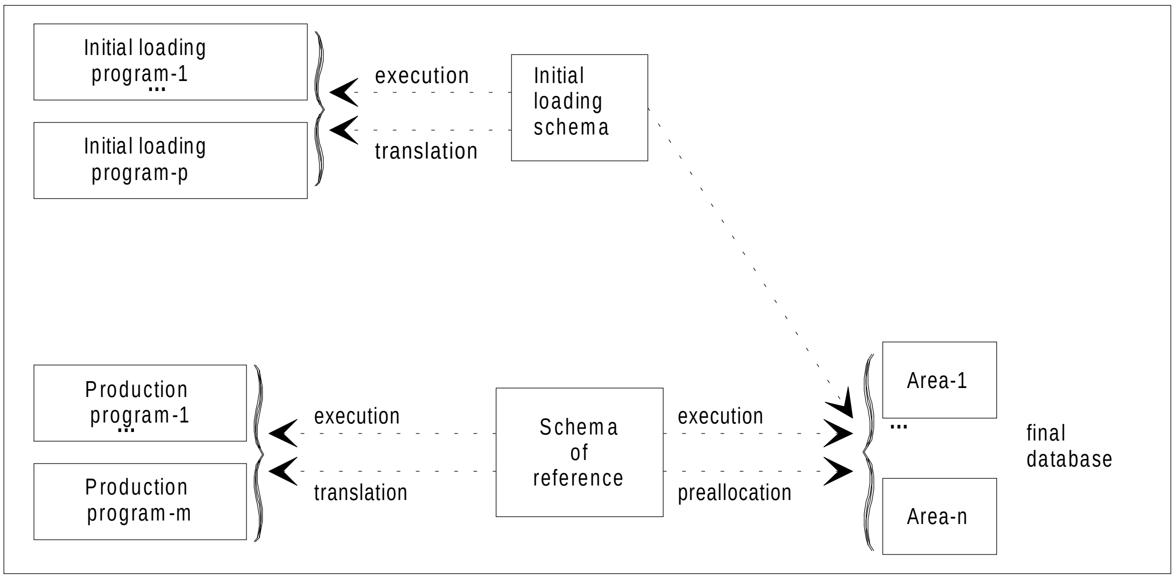


Figure 8-11. Initial Loading Schema

The DDL parameters which can be modified to create a new schema are discussed in the **IDS/II Database Administrator's Guide**.

In the new schema description, the DMCL must be the same. The DDL reference date-time of the loading schema may be different from that of the reference schema. The DMCL values for the reference date-time from the reference schema must be imposed on the loading schema.

The logic of the **initial loading** programs may be different from that of the **production** programs:

1) if the set selection paths are different in the two schemas.

2) if the indexes are **detached** in the initial loading phase and **attached** in the production schema.

   In the second case, the following operations must be performed:

   a) use the BUILD_KEY command from DBUTILITY to load the indexes after the initial loading phase
   b) modify the DMCL to suppress the clause NO ON-LINE UPDATE MANDATORY
   c) modify the DMCL reference date-time

The database areas must be preallocated using the reference schema.

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 Full IDS/II User's Guide Database Products: Full IDS-II

Reference N° :	47 A2 07UDA00	**Date:**	**September 1991**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME : ______________________________ Date : __________

COMPANY : ______________________________

ADDRESS : ______________________________

____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation D$^{ept.}$
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC		
357 AVENUE PATTON	**Phone:**	+33 (0) 2 41 73 72 66
B.P.20845	**FAX:**	+33 (0) 2 41 73 70 66
49008 ANGERS CEDEX 01	**E-Mail:**	srv.Duplicopy@bull.net
FRANCE		

CEDOC Reference #	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		

[_ _] : The latest revision will be provided if no revision number is given.

NAME: ______________________________ Date:__________

COMPANY:______________________________

ADDRESS: ______________________________

____

PHONE: ____________________ FAX: ____________________

E-MAIL: ______________________________

For Bull Subsidiaries:

Identification: ______________________________

For Bull Affiliated Customers:

Customer Code: ______________________________

For Bull Internal Customers:

Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 07UDA00