Full IDS/II

Reference Manual Vol. 2

DPS7000/XTA
NOVASCALE 7000

Database Products: Full IDS-II

REFERENCE
47 A2 06UDB00

DPS7000/XTA NOVASCALE 7000

Full IDS/II

Reference Manual Vol. 2

Database Products: Full IDS-II

February 1992

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

REFERENCE
47 A2 06UDB00

The following copyright notice protects this book under Copyright laws which prohibit such actions as, but not limited to, copying, distributing, modifying, and making derivative works.

Copyright © Bull SAS 1992

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

Intel® and Itanium® are registered trademarks of Intel Corporation.

Windows® and Microsoft® software are registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark in the United States of America and other countries licensed exclusively through the Open Group.

Linux® is a registered trademark of Linus Torvalds.

The information in this document is subject to change without notice. Bull will not be liable for errors contained herein, or for incidental or consequential damages in connection with the use of this material.

Preface

SCOPE AND OBJECTIVES

This version of the Full IDS/II Reference Manual consists of two volumes containing information on the concepts, processors and languages necessary for the efficient operation of an IDS/II database running under GCOS 7. This is the second volume. It is to be used as a reference manual for the Database Administrator or any person having the responsibility of organizing or administrating an IDS/II database.

INTENDED READERS

The two companion volumes of this manual are to be used by persons having the responsibility of organizing or administrating an IDS/II database. Those persons can refer to these two volumes for concise information on IDS/II concepts, programming information, and specific syntax details of the various languages documented herein.

STRUCTURE OF THIS DOCUMENT

This volume of the Full IDS/II Reference Manual describes the Cobol Data Manipulation language and its use directly within COBOL programs. The use of the Subschema, which provides further flexibility in the operation of IDS/II database, is also fully described.

There are six sections in the volume.

Section 1 is a general introduction to the functions of Version 5 of the IDS/II database.

Section 2 describes the communication zones within the IDS/II database: the User Working Area, Special Registers and Currency Indicators.

Section 3 describes the rules governing the use of the subschema.

Section 4 contains a list of the Data Manipulation Language (DML) verbs and their corresponding syntax and usage rules.

Section 5 gives the necessary information concerning the compiling and linking of the Cobol/DML source.

Section 6 describes the details behind the execution of the COBOL/DML program once linkage and compilation have been successfully completed.

RELATED DOCUMENTS

The manuals below provide the information necessary for the use of an IDS/II database in the GCOS 7 environment.

Full IDS/II Reference Manual Volume 1 ..... *47A205UD*
Full IDS/II User's Guide *47A207UD*
IDS/II Administrator's Guide *47A213UD*
Database Reorganization Utility User's Guide *47A215UD*
COBOL 85 Reference Manual *47A205UL*
COBOL 85 User's Guide *47A206UL*

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: **SCHEMA** 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 special symbols, the following 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; at most 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 database management subsystem which runs on the GCOS7 operating system; it provides many independent users with an access to an integrated database. IDS/II provides each user with a separate subset view of the total database structure. The entire logical structure of the database is described by the Schema Data Description Language (DDL). The physical characteristics are expressed by the Schema Device Media Control Language (DMCL). Separate views are defined by the Subschema Data Description Language (SDDL).

The Database Administrator (DBA) of an IDS/II database is a person or group of persons who control the design, creation, access controls and the use of the Schema description and database files. Once the database design has been successfully translated by the appropriate function, then Subschema views can be prepared which will provide the user with the separate view of the database which interests him. It is the DBA who determines the scope of access to any Subschema created within the IDS/II database.

1.1 THE SUBSCHEMA

A Subschema describes the portion of the database which is available to a program. Subschemas are written in the Subschema Data Description Language (SDDL), which is syntactically similar to COBOL. Any consistent part of the Schema Description can be omitted from a subschema; in other words, any areas, sets, records, or data items defined in the schema can be omitted. Thus, only the relevant portions of the database which are needed by a program or a set of programs will be made available using the Subschema Description. Privacy and the integrity of sensitive information is thus ensured because items in the Schema Description which are not specifically required by an application program are not included in a Subschema Description. The DBA controls the creation and use of the schema, the validation of any subschemas, and the overall contents of the database.

After a subschema has been successfully translated by the Subschema Translator Function, the resultant subschema must be validated against the specified schema. In this process, names contained in the subschema are checked against their corresponding names in the schema. Realms, sets, records and data items can be included in the subschema; they can each be renamed in the ALIAS Section of the subschema to allow the program to retain any unique naming conventions not contained in the Schema Description. The validation process determines what operations are permitted against the database; for example, which records can be located, stored, erased or modified in terms of the presence or absence of control information.

1.2 DATA TRANSFORMATIONS

The validation process determines what transformations are needed to map a field description in the schema to a different description in the subschema. This is necessary because data formats can differ between the schema and the subschema.

1.3 DATA INDEPENDENCE

A program can remain independent of many types of changes to IDS/II database formats. This independence is due to two factors: the data transformation capability and the independence provided by the subschema view. Structural changes in DDL entities do not affect the subschema view. Such structural changes are taken into account when the validation process is applied to the subschema to determine any new mapping rules (data transformations and internal DBCS code correspondences).

1.4 COMPILATION

When a subschema is validated, IDS/II produces a series of reports which provide a complete description of all COBOL Data Manipulation Language functions that are permitted or prohibited for that subschema. References to any names which have been omitted from the subschema are prohibited, since they are undefined to the program. Certain DML functions, such as FIND, MODIFY and STORE can be restricted if certain control information is not included in the subschema. For example, if a set is sorted on a control field within a member record, that field must be included in the record description within the Subschema Description. Otherwise, the program is unable to CONNECT this record.

An attempt to compile a program using a DML statement which IDS/II has prohibited in the validation reports may result in a diagnostic indicating that a prohibited DML function has been used. Therefore, it is the responsibility of the DBA to communicate a list of DML functions which are permitted for a given subschema. At execution time, IDS/II runs a check to ensure that each DML action requested (for example, CONNECT) is valid for that subschema. An error exception condition is raised if an attempt is made to use a prohibited DML function.

1.5 USER WORKING AREA

The COBOL compiler creates a User Working Area (UWA) which contains all the data items and database parameters that are required by the program. The UWA is analogous to a WORKING-STORAGE or LINKAGE Section specified by the subschema and reserved for IDS/II data items. Any data items included in the subschema are available for use by the program. Any database parameters required are also included in the UWA.

For example, parameters can be specified in the schema for a LOCATION MODE IS DIRECT record, an AREA-ID parameter, or within a SET SELECTION clause. These parameters represent fields required by the Database Control System (DBCS) to obtain access to the database. It is the responsibility of the DBA to communicate these parameters to the programmer. Proper use of the parameters, their names, and their formats are the responsibility of the user.

1.6 DATA MANIPULATION STATEMENTS

The Data Manipulation Language (DML) offers a complete set of statements to be used within a COBOL program. These statements are directed to the following entities: realms, records, sets, keys and fields.

Each realm which is to be used must be opened before any data within it can be accessed. Once access to that realm is completed, it must be closed to prevent further access.

Records are located by first moving any required control data values into the appropriate control fields in the UWA. The program then issues a FIND statement specifying which method must be used to locate the record. Once found, the data in the record can be transformed into the subschema formats by a GET statement. Finally, a new record occurrence is added to the database by a STORE statement, once all the data necessary for that record are established in the UWA. An existing record can be removed by an ERASE statement. Any record can be accessed and contents of the fields in that record can be modified to alter data in the database.

In certain cases, if control fields are changed, the relationship of the record to its membership in sets can be altered. An existing record can be placed into a MANUAL set by the CONNECT statement or removed from an OPTIONAL set by the DISCONNECT statement.

Status information which will provide database key values and realm names for current records to the program can be obtained from the system.

1.7 REALM FILES

In the schema, storage **areas** are referred to as areas; in the subschema, storage areas are referred to as **realms**. Each refers to the same thing: a storage area. Therefore, all

references to the word **realm** concern the subschema and the COBOL program view of an area.

1.8 INDEX FILES

The KEY facility is provided to the COBOL programmer through index files which contain entries defined in the Schema DMCL. When using keys, the programmer must assign these index files, but READY verbs must not be assigned to them.

1.9 ACCESS RESTRICTION

The DBA can specify access restrictions on portions of the database viewed through the subschema. These restrictions prohibit the application from issuing DML requests against specified records and/or realms. An example of a possible access restriction would be a program being able to retrieve a record but unable to modify or erase it.

Access restrictions can be directed to realms, records, sets or members of sets. These restrictions are described below:

- **Restrictions on realms:** these prevent a program from readying a realm in a specified usage-mode. If a subschema contains an update access restriction on a realm, programs using this subschema will not be allowed to ready the realm in update mode and will thus be unable to update records in this realm.

- **Restrictions on records:** these prevent a program from performing one or more of the following DML verbs against the specified record-types or fields: ERASE, FIND, GET, MODIFY, STORE.

- **Restrictions on sets:** these prevent a program from issuing one or more of the following DML verbs against record occurrences within the specified set-types: CONNECT, DISCONNECT, FIND.

- **Restrictions on members:** these prevent a program from performing one or more of the following DML verbs against the specified members of a set: CONNECT, DISCONNECT, FIND.

NOTE: Access restrictions always concern the record, realm or set within its role as object of the specified DML verb and not the database items themselves with all their possible relationships. For example, an access restriction prohibiting the FIND verb for a record will not prohibit access to this record through the set selection mechanism.

1.10 UTILITIES

The DBA has available to him a number of utilities for managing the physical realm files where the database is stored. The utilities are described below:

- The Print Utility: prints portions of a realm (record occurrences or entire pages).
- The Analyze Utility: analyzes an entire realm file or portions of a realm file to provide reports on record occurrences and space utilization; also produces graphs representing space utilization and access performance.
- The Validate Utility: validates the logical structure of realm files and repairs destroyed pages.
- The Build Utility: builds or re-builds (deletes) an index file containing key entries.
- The Generate Utility: generates a new database from an existing one after changes on physical or logical structures (database reorganization).

2. Communication Zones

The Data Manipulation Facility is the term applied to the group of mechanisms which directly access the IDS/II database. This access facility is composed of three communication zones which exist between the user and the Database Control System (DCBS). These three zones are the User Working Area records, Special Registers and Currency Indicators. Each is fully described in the following subsections.

2.1 USER WORKING AREA

The User Working Area (UWA) is a temporary storage area where record occurrences are placed before storage in the database itself or after retrieval from it. The programmer need not define the UWA zones corresponding to the subschema records because they are derived automatically from the subschema by the compiler and made available to the program in the form of declarative statements in COBOL format.

2.2 DB-KEY DATA ITEMS

In the WORKING-STORAGE or LINKAGE SECTION, data items containing data-base-keys are defined with the type USAGE IS DB-KEY:

```
level-number identifier USAGE IS DB-KEY.
```

This type of data description is required for the "identifier" of the following DML statements:

```
ACCEPT identifier FROM ... CURRENCY
ACCEPT ... FROM  identifier REALM-NAME

                             {NEXT }
ACCEPT identifier FROM ... {PRIOR}
                             {OWNER}
ACCEPT identifier FROM ... MINIMUM-DB-KEY
FIND ... DB-KEY IS identifier
```

USAGE DB-KEY is equivalent to USAGE COMP-2 when these data items are used in COBOL 74 statements.

2.3 DATABASE SPECIAL REGISTERS

Database special registers are areas used to communicate database information between the program and the Database Control System. The program can read the contents of the registers, but it cannot alter their contents. The following reserved words can be placed in special registers, and are described in the subsections below:

```
<DB-STATUS, DB-REALM-NAME, DB-RECORD-NAME, DB-SET-NAME, DB-KEY-NAME, DB-KEY-NAME and DB-
DETAILED-STATUS.
```

2.3.1 DB-STATUS

The reserved word, DB-STATUS, returns the status of every DML statement at the conclusion of its execution. The implicit description of the return is an alphanumeric data item of seven characters. If no exception is encountered during the execution of the statement, DB-STATUS is set to zeroes. If an exception is encountered, DB-STATUS is set to the appropriate value. For further information, see the subsection entitled "Database Status Indicators", below.

2.3.2 DB-REALM-NAME

The reserved word, DB-REALM-NAME, returns the name of a realm at the conclusion of relevant DML statements. The implicit description of the return is an alphanumeric data item of thirty characters. A successful statement (other than FIND or STORE) leaves this register unchanged. Each successful FIND and STORE updates DB-REALM-NAME with the appropriate realm-name. An unsuccessful statement may supply a realm-name in the register in addition to the data-base-exception code supplied in DB-STATUS. If a realm-name is not relevant to the unsuccessful function, the register is set to blanks.

2.3.3 DB-RECORD-NAME

The reserved word, DB-RECORD-NAME, returns the name of a record at the conclusion of relevant DML statements. The implicit description of the return is an alphanumeric data item of thirty characters. A successful statement (other than FIND or STORE) leaves this register unchanged. Each successful FIND and STORE updates DB-RECORD-NAME with the appropriate record-name. An unsuccessful statement may supply a record-name in the register in addition to the data-base-exception code supplied in DB-STATUS. If a record-name is not relevant to the unsuccessful function, the register is set to blanks.

2.3.4 DB-SET-NAME

The reserved word, DB-SET-NAME, returns the name of a set at the conclusion of relevant DML statements. The implicit description of the return is an alphanumeric data item of thirty characters. A successful statement leaves this register unchanged. An unsuccessful statement may supply a set-name in the register in addition to the data-base-exception code supplied in DB-STATUS. If a set-name is not relevant to the unsuccessful function, the register is set to blanks.

2.3.5 DB-KEY-NAME

The reserved word, DB-KEY-NAME, returns the name of a key at the conclusion of relevant DML statements. The implicit description of the return is an alphanumeric data item of thirty characters. A successful statement leaves the register unchanged. An unsuccessful statement may supply a key-name in the register in addition to the data-base-exception code supplied in DB-STATUS. If a key-name is not relevant to the unsuccessful function, the register is set to blanks.

2.3.6 DB-DETAILED-STATUS

The reserved word, DB-DETAILED-STATUS, returns an explanatory message when an abnormal conclusion of certain DML statements occurs.

The implicit description of the return is the following:

```
01 DB-DETAILED-STATUS.
02 DB-MESSAGE-LENGHT  USAGE COMP-1
02 DB-MESSAGE-TEXT    PIC LX(256)
                      DEPENDING ON DB-MESSAGE-LENGHT.
```

If no message is present, DB-MESSAGE-LENGTH is set to 0.

2.4 DATABASE STATUS INDICATORS

The execution of any database manipulation statement produces a value in the special register DB-STATUS. This value, known as a **database status indicator**, consists of a statement code in the leftmost two character positions and a status code in the rightmost five character positions. If the execution of any database manipulation statement results in a data-base-exception, the statement code indicates which type of manipulation statement caused the exception. Whenever the execution of any database manipulation statement does not result in a data-base-exception condition, both the statement code and the status code are set to zero.

2.4.1 Statement Code and Status Code

The leftmost two character positions of the database status indicator contain the DML statement codes described in Figure 2-1.

Table 2-1. Statement Codes

Statement	Statement Code
ACCEPT	01
CONNECT	02
DISCONNECT	03
ERASE	04
FIND	05
FINISH	06
GET	08
DB Condition (IF)	09
MODIFITY	11
READY	13
STORE	15

The rightmost five character positions of the database status indicator contain a status code which indicates a specific data-base-exception condition as described in Figure 2-2.

Table 2-2. Database Status Codes (1/2)

Status Code	Data-base-exception Condition
(DATA BASE RETRIEVAL EXCEPTION CONDITIONS)	
02100	A prior record has been requested, but no record of any type declared in the subschema exists before the current position in the set, realm or key.
	A next or duplicate record has been requested, but no record of any type declared in the subschema exists after the current position in the set, realm or key.
02200	An unavailable record is required by the DBCS.
02300	No set can be located to satisfy the set selection or set ordering criteria.
02400	No record can be located to satisfy the record-selection-expression.
(CURRENCY INDICATOR EXCEPTION CONDITIONS)	
03100	Current record of realm, set-type, key-type or record-type is null (or virtual).
03200	Current record of the run-unit is null.
03300	Current record of the run-unit, realm, set-type or key-type is not of the correct record-type.
(NAME SPECIFICATIONEXEPTIO N CONDITIONS)	
04100	Data-base-key value is inconsistent with realm-name.
04200	The subschema does not include a data item set-type required by set selection or set ordering criteria.
04300	Realm-name is invalid (not known to the subschema or inconsistent with record-type).
(DATA ITEM VALUE EXCEPTION CONDITIONS)	
05100	Contents of data item are duplicated in the database.
05200	Contents of data item do not satisfy validity checks.
05300	Transformation of data item violates a data transformation rule.
(DELETED RECORD EXCEPTION CONDITIONS)	
07200	Deletion of non-empty set (ERASE).
(MEMBERSHIP	EXCEPTION CONDITIONS)
08100	Object record is already connected to the set.
08300	Object record is not connected to the set.
08500	Set membership change is required.
(READY MODE EXCEPTION CONDITIONS)	
09100	Realm not in ready state or database not open.

09200	Realm not in update mode.
09300	Realm already in ready state.

Table 2-2. Database Status Codes (2/2)

Status Code	**Data-base-exception Condition**
(GCOS7 SPECIFIC)	
70200	Index resource not available.
70434	Realm not assigned.
70435	External file name unknown.
70436	Usage-mode conflict with ASSIGN/DEFINE
73615	Database/Index inconsistency.
73630	Ordinal null in FIND identifier WITHIN set or realm.
73640	Data item modification cannot be performed without migration.
73650	Set membership violates AREA of OWNER clause.
73660	UWA record-description missing.
75390	Realm or index not assigned.
75391	EFN unknown
75392	Usage mode conflict with ASSIGN/DEFINE
(RESOURCE	ALLOCATION EXCEPTION CONDITIONS)
80200	Space in realm is exhausted.
80300	Space in index resource is exhausted.
(ACCESS CONTROL LOCK EXCEPTION CONDITIONS)	
90100	Access restriction is specified.

2.4.2 Statement Code and Status Code Combinations

The valid possible combinations of statement code and status code are shown in Figure 2-3 which is actually a list of the generic DML functions followed by the relevant code. An "X" indicates a valid combination of status code and statement code. Detailed information on these functions can be found in the description of each DML statement in Section 3.

Table 2-3. Statement and Status Code Combinations

STATUS CODE	**STATEMENT CODE**										
	ACCEPT	CONNECT	DISCONNECT	ERASE	FIND	FINISH	GET	IF	MODIFY	READY	STORE
	01	02	03	04	05	06	08	09	11	13	15
02100	X				X						
02200	X	X	X	X	X			X	X		X
02300		X			X				X		X
02400					X						
03100	X				X			X			
03200	X	X	X	X	X		X	X	X		
03300		X	X	X	X		X	X	X		
04100	X				X						X
04200		X			X				X		X
04300					X						X
05100		X							X		X
05200					X				X		X
05300					X		X		X		X
07200				X							
08100		X									
08300			X						X		
08500									X		
09100	X	X	X	X	X	X	X	X	X		X
09200		X	X	X					X		X
09300										X	
70200	X			X	X				X		X
73590										X	
73591										X	
73592										X	
73615	X			X	X				X		X
73630					X						
73640									X		
73650		X							X		
73660					X		X		X		X
80200									X		X
80300									X		X
90100	X	X	X	X	X		X		X	X	X

2.4.3 Use of Special Registers

When the DBCS recognizes certain data-base-exception conditions, it places the following into special registers whenever relevant:

```
DB-REALM-NAME, DB-RECORD-NAME and DB-DETAILED-STATUS, DB-SET-NAME, DB-KEY-NAME.
```

DB-REALM-NAME

During the execution of a Database Manipulation Statement (DML) that has not completed successfully, the realm-name of the realm associated with the data-base-exception condition is placed in DB-REALM-NAME. For example: FIND NEXT WITHIN realm-name1 can return an "End of Realm" exception, in which case the name of realm-name1 will be placed in DB-REALM-NAME.

DB-RECORD-NAME

During the execution of a DML statement that has not completed successfully, the record-name of a record-type associated with the data-base-exception condition is placed in DB-RECORD-NAME.

DB-SET-NAME

During the execution of a DML statement that has not completed successfully, the set-name of the set-type associated with the data-base-exception condition is placed in DB-SET-NAME. For example: FIND NEXT WITHIN set-name1 can return an "End of SET" exception, in which case the set-name of set-name1 will be placed in DB-SET-NAME.

DB-KEY-NAME

During the execution of a DML statement that has not completed successfully, the key-name of the key-type associated with the data-base-exception condition is placed in DB-KEY-NAME. For example: FIND NEXT WITHIN key-name1 can return an "End of Key" exception, in which case the key-name of key-name1 will be placed in DB-KEY-NAME.

DB-DETAILED-STATUS

During the execution of a DML statement that has not successfully completed, an explanatory message may appear in DB-DETAILED-STATUS giving further information concerning the DB-STATUS value. For example, a STORE statement may return a DB-DETAILED-STATUS message indicating which field did not satisfy the validity checks.

2.5 DATABASE CURRENCY INDICATORS

In the IDS/II database environnement, the user must be able to access records that are a part of more than one logical relationship. Unlike a database built on sequential files, in IDS/II many records can logically follow a given record, depending upon the structure of the database and the search criteria in use. Thus, there can be several simultaneous "current records" during the execution of a run-unit, and several simultaneous "current record pointers". These simultaneous current record pointers are known as "currency indicators". The following currency indicators are maintained during the execution of a program:

- the current record of the run-unit.
- the current record of each set-type.
- the current record of each realm.
- the current record of each record-type.
- the current record of each key-type.

The DBCS maintains a pointer to each of these record occurrences. Any of these pointers may be NULL; that is, they may specify no record if there is no current record of that type. The current record of a set-type, of a realm or of a key-type may be "virtual" if it identifies a position in a set, a realm or a key that is located between two records. This can happen because of a DISCONNECT or ERASE statement, when there is no longer a record to point to, but a virtual space instead. A virtual currency indicator is valid for all DML statements except "FIND CURRENT WITHIN ...", "FIND DUPLICATE WITHIN ...", "ACCEPT ... FROM set-name, key-name or realm-name CURRENCY or REALM-NAME", and "Data Base Condition (IF)".

2.5.1 Current Record of the Run-unit

The pointer for each run-unit is maintained to identify the object record for certain DML statements that do not specify a particular record. The execution of some types of statements causes the DBCS to update its pointer to a different record, while the execution of other types of statements does not affect the pointer to the current record of the run-unit.

2.5.2 Current Record of a Set-type

For each Set Description Entry, the DBCS maintains a separate pointer. The pointer for each set-type is maintained by the DBCS to identify the record in the set-type that was last referenced by a successfully executed statement. A single record, when accessed, sets the currency indicators for all sets in which it is an owner or member, unless a RETAINING phrase was specified in the DML statement. A DISCONNECT or ERASE statement may set the currency indicators to the position between two records that the current record occupied.

2.5.3 Current Record of a Realm

For each Realm Description Entry, the DBCS maintains a separate pointer. The pointer for each realm is maintained by the DBCS to identify the record in that realm that was last referenced by a successfully executed statement, unless a RETAINING phrase was specified in the DML statement. An ERASE statement may set the currency indicators to the position between two records that the current record occupied.

2.5.4 Current Record of a Record-type

For each Record Description Entry, the DBCS maintains a separate pointer. The pointer for each record-type is maintained by the DBCS to identify the record of that type that was last referenced by a successfully executed statement, unless a RETAINING phrase was specified in the DML statement.

2.5.5 Current Record of a Key-type

For each Key Description Entry, the DBCS maintains a separate pointer. The pointer for each key-type is maintained by the DBCS to identify the record in that key-type that was last referenced by a successfully executed statement. A single record, when accessed, sets the currency indicators for all keys in which it is defined, unless a RETAINING phrase was specified in the DML statement. An ERASE statement may set the currency indicators to the position between two records that the current record occupied.

2.5.6 Currency Values

In addition to currency indicators, the user can keep a trace of the data-base-key value for any record or set accessed, as explained in the example. A statement such as

```
ACCEPT SAVE-RECORD-03-DBK FROM RECORD-03 CURRENCY.
```

would allow the manual retention of the location for the current record of that type. In some complex structures, it may be advisable to keep needed currency values within the control of the program, since many DML functions can change the currency indicators. When the record in the example is needed later, a statement such as

```
FIND RECORD-03 DB-KEY IS SAVE-RECORD-03-DBK.
```

would re-establish the saved record as current-of-run-unit and reset all appropriate currency indicators for the record.

2.5.7 Currency Indicators and DML Statements

The Table 2-1 shows each DML statement with respect to each of the four types of currency indicators. It indicates the currencies which are used as implicit input parameters and those which are updated on successful completion.

Table 2-4. Currency Usage and Update (1/4)

STATEMENT	CURRENCIES USED IN INPUT	CURRENCIES UPDATE ON SUCCESSFUL COMPLETION
ACCEPT... FROM CURRENCY	current-of-run-unit	
ACCEPT... FROM realm-n CURRENCY	current-of-realm specified	
ACCEPT... FROM record-n CURRENCY	current-of-record-type specified	
ACCEPT... FROM set-name CURRENCY	current-of-set-type specified	
ACCEPT... FROM key-name CURRENCY	current-of-key-type specified	
ACCEPT... FROM REAL-NAME	current-of-unit	
ACCEPT... FROM record-name REALM-NAME	current-of-record-type specified	
ACCEPT.. FROM set-name REALM-NAME	current-of-set-type specified	
ACCEPT.. FROM key-name REALM-NAME	current-of-key-type specified	
ACCEPT... FROM ident REALM-NAME		
ACCEPT... FROM set-nsame NEXT/ PRIOR/OWNER	current-of-set-type specified	
CONNECT... TO set-name	-current-of-run-unit as object record -if first level of set selection path is identified by APPLICATION, the corresponding current of set-type is used. In addition, if the set selection path has only one level and insertion is NEXT or PRIOR, the current-of-set-type identifies the insertion point	current-of-set-type specified, if not retained

Table 2-4. Currency Usage and Update (2/4)

STATEMENT	CURRENCIES USED IN INPUT	CURRENCIES UPDATE ON SUCCESSFUL COMPLETION
DISCONNECT FROM set-name	current-of-run-unit as object record	current-of-set-type specified is made "virtual" if the object record was current of this set
ERASE... ERASE... ALL MEMBERS ERASE... SELECTIVE MEMBERS ERASE... PARMANENT MEMBERS	current-of-run-unit as object record	-current-of-run-unit nulled -current-of-realm is made "virtual" if an erased record was current of realm -current-of-record-type is nulled if an erased record was current of its record-type -current-of-set-type of the sets in which an erased record was owner is nulled if this record was the current-of-set -current-of-set-type of the sets in which an erased or disconnected record was an actual member is made "virtual" if record was the current of the set -current-of-key-type of the key in which an erased record participated is made "virtual"
FIND... DB-KEY..		-current-of-run-unit
FIND ANY record-n		-current-of-realm,if not retained -current-of-record-type
FIND DUPLICATE	current-of-run-unit, to identify CALC record-type and position in "synonym" chain	-current-of-set/key-type, if not retained, for all sets/keys in which the record participates (and those to which the record is connected for set manual)
FIND ANY [rec-n]		
USING key-name		
FIND DUPLICATE	current-of key-type specified to identify position in key	
[record-name]		

Table 2-4. Currency Usage and Update (3/4)

STATEMENT	CURRENCIES USED IN INPUT	CURRENCIES UPDATE ON SUCCESSFUL COMPLETION
FIND FIRST/LAST ordinal... WITHIN realm-name		current-of-set-type specified to identify set occurence
FIND FIRST/LAST ordinal... WITHIN key-name		
FIND NEXT/PRIOR WITHIN set-name	current-of-set-type specified,to identify set occurence and position in set occurrence	
FIND NEXT/PRIOR WITHIN key-name	current-of-key-type specified,to identify position in key	
FIND OWNER WITHIN set-name	current-of-set-type specified to identify set occurrence	
FIND CURRENT	current-of-run-unit	
FIND CURRENT record-name	current-of-record-type specified	
FIND CURRENT WITHIN realm-name	current-of-realm specified	
FIND CURRENT WITHIN set-name	current-of-set-type specified	
FIND record-name WITHIN set name CURRENT [USING...]	current-of-set-type specified, to identify the set occurrence	
FIND record-name WITHIN set-name [USING...]	if first level of set selection is by APPLICATION, the corresponding current-set-type is used	
FIND DUPLICATE WITHIN set-name USING...	current-of-set-type specified, to identify record-type, set occurrence and position in set occurrence	

Table 2-4. Currency Usage and Update (4/4)

STATEMENT	CURRENCIES USED IN INPUT	CURRENCIES UPDATE ON SUCCESSFUL COMPLETION
FINISH...		
GET	current-of-run-unit as object record	
OWNER/MEMBER/ TENANT condition	current-of-run-unit-as object record	
Membership condition	current-of-set-type as object record	
MODIFY... (no membership)	-current-of-run-unit, as object record	-current-of-realm, if not retained
MODIFY... MEMBERSHIP	-current-of-run-unit,as object record -if first level of set selection path of a new set occurrence identified by APPLICATION, the corresponding current-of-set is used. If the set selection path has only one level and ORDER is NEXT or PRIOR, the current-of-set-type identifies insertion point	-current-of-record, if not retained -current-of-set-type,if not retained, of all the sets in which the records is a new owner or a member currently connected. -current-of-key-type, if not retained,of all the keys in which the record participates -current-of-run-unit in case of migration (only the db-key is modified)
READY...		
STORE record-name	If first level of set selection path of a new set occurrence identified by APPLICATION, the corresponding current-of-set is use	-current -of-run-unit -current-ofrealm, if not retained -current-of-record-type, if not retained -current-of-set-type, if not retained, of all the sets in which the record is an owner or a member currently connected -current-of-key-type, if not retained, of all the keys in which the record participates
IDS session initial conditions		all currencies are nulled

3. Subschema Usage

3.1 SCHEMA/SUBSCHEMA DATA CONVERSION

The representation of a data item as it appears in the database (determined by the Schema DDL) may differ from the representation of the data item in a program's UWA (determined by the subschema).

If these two representations of a data item differ, conversions occur at run-time the following cases:

- a conversion from the database representation to the UWA representation when the value of the data item is required by the program
- a conversion from the UWA representation to the database representation whenever the data item's value is stored or updated
- a conversion from one of these two representations to the other whenever a comparison is required between UWA values and database values.

These conversions are intended to change the representation of the value, not the database value itself, so as to preserve the meaning of the value. For this reason, database-exceptions may be raised when the conversion is not legal. An example of a legal conversion: from numeric data in the database to COMPUTATIONAL format in COBOL Working-Storage.

3.1.1 Data Formats

Within the subschema, the data formats available are those of COBOL. There is a direct valid correspondence between certain schema and subschema formats, as shown in Figure 3-1.

Table 3-1. Equivalence between Schema Data Formats and Subschema Data Formats

Data Format	Schema Format	Subschema COBOL-like Format
Characters	TYPE IS CHARACTER n	PIC X(n) USAGE IS DISPLAY
Packed Decimal	TYPE IS DECIMAL n	PIC 9(n) USAGE COMPUTATIONAL
Unpacked Decimal	TYPE IS UNPACKED	PIC 9(n) USAGE IS DISPLAY
Half Word Binary	TYPE IS BINARY 15	COMPUTATIONAL-1
Full Word Binary	TYPE IS BINARY 31	COMPUTATIONAL-2

During transformation, the content of each item will be checked for proper characters, signs and scale factors. Invalid contents in an item will result in a data-base-exception.

NOTE: Contents of data do not satisfy validity checks.

3.1.2 Elementary Data Transformation Rules

In addition to the direct correspondences listed in Table 3-1, other correspondences are possible. Each identifier in the subschema is matched by name with a corresponding identifier in the schema. Once this match has been established, the appropriate data transformation is executed by the DBCS, whenever possible.

Figure 3-2 provides a list of legal system default transformations and of the system default limitations which are imposed on data values.

Table 3-2. Data Transformation Matrix

SUBSCHEMA DATA TYPE		**SCHEMA DATA TYPE**					
Usage	Picture	FB15	FB31	UNSIGNED DEC (UPK, PK or PK2)	SIGNED DEC (UPK or PK)	CHAR	CHAR dep. on
COMP-1	NONE	YES	D1	D1 D4	D1 D4	/////////	
COMP-2	NONE	E1	YES	E1 E2 E3	E1 E3		
DISPLAY	9.V9.	D1 D2 D3		D1 D3 D4	D1 D2 D3 D4		
COMP	9.V9.						
COMP-8	9.V9	E1 E4		E1 E3 E4	E1 E3 E4		
DISPLAY	S9.V9.	D1 D3		D1 D3 D4	D1 D3 D4		
COMP	S9.V9.	E1 E4		E1 E2 E3 E4	E1 E3 E4		
DISPLAY	X	//////////////////////				D5 D6	
DISPLAY	LX(.)					E5 E6	
YES - Allowable transformation /// - Prohibited transformation							

Meaning of entries:

YES	Allowable transformation, no restriction
///	Prohibited transformation
Ex	Specifies a restriction "x" when encoding (subschema -> schema)
Dx	Specifies a restriction "x" when decoding (schema -> subschema)

where "x" represents one of the following restrictions:

1. If the value of the source data item (after any necessary rounding) exceeds the capacity of the target data item, the transformation is not performed and a data-base-exception occurs.

2. If the value of the source data item is negative, the transformation is not performed and a data-base-exception occurs.

3. If the scale factor of the source data item is less than the scale factor of the target data item, the target data item is filled on the right with zeroes.

4. If the scale factor of the source data item exceeds the scale factor of the target data item, the least significant digits are lost. If these least significant digits are only

zeroes, the transformation is successful. If truncation of non-zero digits occurs during decoding, the transformation is performed and a data-base-exception occurs. If the truncation occurs during encoding, the transformation is not performed and a data-base-exception occurs.

5. If the length of the source data item is less than the length of the target data item, the target data item is filled on the right with blank characters. When the target data item is of varying length, its length is evaluated after having removed all trailing blank characters in the source data item. In this case, there will be no right padding with blank characters.

6. If the length of the source data item exceeds the length of the target data item, the value of the source data item is truncated to the length of the target data item. If only blank characters are lost, the transformation is successful. If non-blank characters are lost during decoding, the transformation is performed and a data-base-exception occurs. If the truncation occurs during encoding, the transformation is not performed and a data-base-exception occurs.

3.2 SUBSCHEMA SECTION IN DATA DIVISION

3.2.1 Function

The Subschema Section, which must be the first section of the Data Division, contains only one entry, the SUBSCHEMA (DB) Entry. This entry specifies which database subschema descriptions are to be accessed by the program and offers the means of unambiguous qualification in case of the same name exists for several subschemas.

More than one DB entry can appear in a given program.

3.2.2 General Format

```
DATA DIVISION.

SUB-SCHEMA-SECTION.

DB db-internal-name USING subschema-name WITHIN schema-name.

[                  {WORKING-STORAGE}           ]
[DB-DESCRIPTIONS in {                } SECTION .]
[                  {LINKAGE         }           ]
```

3.2.3 Rules

1. The subschema referenced by subschema-name within schema-name must be made available to the program during execution For further information, see the subsection entitled "User program execution".

2. The DB-DESCRIPTIONS clause specifies where the subschema record descriptions and DBCS control structures are to be placed when the program is compiled.

3. The *db-internal-name* is an IDS/II reference used at compile time which may be ambiguous, for instance, if the same record name exists between two subschemas. If the program is a MAIN program (not called by another DML program) of a batch step or of a TDS transaction processing routine, the DB-DESCRIPTIONS clause must specify WORKING-STORAGE. If the entire clause is omitted, WORKING-STORAGE is the default value.

4. If the program is a SECONDARY program (called by another program retrieving the same subschema), the DB-DESCRIPTIONS clause must specify LINKAGE SECTION. A secondary program may call another secondary program.

5. Each call to a secondary program must contain in the USING phrase of the CALL arguments, in the USING phase of the CALL statement, concerning the data-base-identifiers that will be used in the execution of the secondary program.

   In a secondary program, the USING phrase of the PROCEDURE statement must contain similar arguments concerning the data-base-identifiers in the same order. It is the user's responsability to provide this argument list. The argument list must be defined as follows:

   ```
   USING {ua} ... DB-REGISTERS DB-CXT [DB-PARAMETERS]    {rn} ...
   ```

   in which the parameters are defined as follows:

   - ua ... represents the user's data arguments that are passed between programs.

   - DB-REGISTERS, DB-CXT and DB-PARAMETERS pass the necessary control structures to permit the execution of all DML functions.

NOTE: DB-PARAMETERS must not be specified if the Schema DDL does not contain any data-base-parameters.

   - rn ... represents the list of record-names referenced in the called program. The record-names must be the same as those recorded in the referenced subschema. If a record-type has no fields, it must not be specified in the list.

3.3 INTER-PROGRAM LINKAGE

To clearly explain inter-program linkage, let us consider a load-module with 3 compile-units, each one corresponding to a particular database access function.

A master program, PM, prepares a single area, AREA-X, of subschema SSDB-L and calls two programs, P1 and P2, which reference the same subschema as PM.

Program P1 references two record-types, RT-A and RT-B, while program P2 references three record-types, RT-A and RT-D and RT-G. The structure of these programs is shown below.

Program PM:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PM.
...
DATA DIVISION.
SUB-SCHEMA SECTION.
DB db-internal-name1 USING SSDB-L WITHIN SDB-L.
...
PROCEDURE DIVISION.
    READY AREA-X ... .
    ..
    CALL P1 USING MYARG DB-REGISTERS DB-CXT
DB-PARAMETERS
                    RT-A RT-B.
    ...
    CALL P2 USING MAX MAY DB-REGISTERS DB-CXT
DB-PARAMETERS
                    RT-A RT-D RT-G.
    ...
    FINISH AREA-X.
    ...
    STOP RUN.
```

Program P1:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. P1.
...
DATA DIVISION.
SUB-SCHEMA SECTION.
DB db-internal-name2 USING SSDB-L WITHIN SDB-L.
         DB-DESCRIPTIONS IN LINKAGE SECTION.
...
PROCEDURE DIVISION USING FUNCX DB-REGISTERS DB-CXT
                              DB-PARAMETERS RT-A RT-B.
         ...
         FIND RT-A ...
         ...
         EXIT PROGRAM.
```

Program P2:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. P2.
...
DATA DIVISION.
SUB-SCHEMA SECTION.
DB db-internal-name3 USING SSDB-L WITHIN SDB-L.
       DB-DESCRIPTIONS IN LINKAGE SECTION.
...
PROCEDURE DIVISION USING ARA ARB DB-REGISTERS DB-CXT
DB-PARAMETERS RT-A RT- RT-GB.
         ...
       FIND RT-D ...
       ...
       ACCEPT MID FROM RT-A CURRENCY.
       ...
ERASE RT-G.
       ...
EXIT PROGRAM.
```

3.4 MULTIPLE SUBSCHEMA FUNCTION

The multiple subschema function allows an application (load-module or transaction) to simultaneously access more than one database at a time by using separate IDS/II sessions known as run-unit sessions.

Each subschema used by these run-unit sessions requires a separate User Working Area and separate groups of currencies.

Two methods exist for the user to build his application, regardless of methodology or limit considerations (such as modularity versus number of compile-units):

Method 1:

The user can handle all database accesses in a single "DML-dedicated" compile-unit, by using several DB-clauses. This may lead to qualification problems between IDS/II entities. See Section 4 for further details.

Method 2:

The user can also divide "n" database accesses among "p" compile-units. In this case the following rules must be applied:

a) **only one program** from the group of "p" compile-units can have the DB-DESCRIPTIONS in WORKING for each of the subschemas which are referenced. Conversely, the other program(s) must have the DB-DESCRIPTIONS in LINKAGE for the relevant subschema. The corresponding DB-CXT, DB-PARAMETERS, DB-REGISTERS and record zones must be passed as parameters.

b) a given compile-unit may be the MAIN program for Subschema S1 (containing the DB-DESCRIPTIONS in WORKING in the relevant DB-clause) and simultaneously be the SECONDARY program for Subschema S2 (containing the DB-DESCRIPTIONS in LINKAGE for this second DB-clause).

NOTE: It is important to take the following restriction into account:

The Multiple Subschema Function does not permit simultaneous access to a database through two different subschemas. However, it is possible to access a database through one subschema, immediately followed by a second database access via another subschema. This implies that the first run-unit session ends before the second run-unit session begins.

This restriction exists because of the possible inconsistency the simultaneous access of two subschemas would have on the database. For example, a current-of-run-unit might be established via one subschema and erased by the other. Or information might be retrieved via one subschema while area page information might be reogranized with DML update verbs via the other subschema.

Violation of this rule will cause an abort (Programming rule violation DDM1001).

4. Data Manipulation Language

The Data Manipulation Language (DML) consists of the statements which are listed and explained in Figure 4-1 below:

```
ACCEPT            - Obtains the data-base-keys or realm-names held within the
                    DBCS currency indicators, or obtains DMCL inFormation from
                    the object schema.

CONNECT           - Inserts a new record occurrence into a set in which the record
                    has been defined with one of the following types of membership:

                         MANUAL MANDATORY
                         MANUAL FIXED
                         MANUAL OPTIONAL
                         AUTOMATIC OPTIONAL

DISCONNECT        - Removes a record from a set in which it resides as an
                    OPTIONAL member.

ERASE             - Removes a record from the database.

FIND              - Locates any record in the database subject to a variety of
                    record-selection-expression options.

FINISH            - Makes a realm unavailable for further access.

GET               - Obtains the contents of a current record.

DB Condition      - Tests database conditions.

MODIFY            - Alters the contents of data items in the database and/or alters
                    the set relationships of a record.

READY             - Makes the contents of a realm available for processing.

STORE             - Adds a new record occurrence to the database.

USE               - Defines a DB-EXCEPTION procedure which can be
                    automatically invoked when needed.
```

Figure 4-1. Data Manipulation Language Verbs

The following pages describe each DML statement in detail. The tables showing the DB-STATUS values in each subsection are to be interpreted using the symbols given below:

- "-" in the REALM, RECORD, SET or KEY column indicates that the corresponding register DB-REALM-NAME, DB-RECORD-NAME, DB-SET-NAME or DB-KEY-NAME is left unchanged by the DBCS.

- A blank in the REALM, RECORD, SET or KEY column indicates that the corresponding register is filled with blanks by the DBCS.

- "X" in the REALM, RECORD, SET or KEY column indicates that the corresponding register is filled by the DBCS with the name of a realm, record, set or key. If "X" is enclosed in parentheses (), the register is filled with either a name or with blanks, depending on the error.

The order in which the error conditions are returned in response to a given DML statement reflects, in most cases, the sequence in which tests were performed by the DBCS during the execution of the DML function.

4.1 ACCEPT

4.1.1 Function

This statement has the following functions:

1. causes the contents of the specified currency indicators to be made available to the program,
2. provides a means for deriving the realm-name which corresponds to a data-base-key value,
3. supplies the data-base-keys of the NEXT, PRIOR, and/or OWNER of a record within a set,
4. reads DMCL inFormation from the object schema.

4.1.2 General Format

Format 1

```
                          [record-name]
ACCEPT identifier-1 FROM  [set-name   ] CURRENCY .
                          [realm-name ]
                          [key-name   ]
```

Format 2

```
                          [record-name ]
ACCEPT identifier-2 FROM  [set-name    ] REALM-NAME .
                          [identifier-3]
                          [key-name    ]
```

Format 3

```
                                    [NEXT ]
ACCEPT identifier-4 FROM set-name   [PRIOR] .
                                    [OWNER]
```

Format 4

```
ACCEPT identifier-5 FROM realm-name LINES-PER-PAGE .
```

Format 5

```
ACCEPT identifier-6 FROM realm-name MINIMUM-DB-KEY .
                                    [OF record-name]
```

Format 6

```
ACCEPT identifier-7 FROM realm-name NUMBER-OF-PAGES .
                                    [OF record-name]
```

Format 7

```
                                   {NEXT }
ACCEPT identifier-8 FROM key-name  {     } .
                                   {PRIOR}
```

4.1.3 Syntax Rules

1. Identifier-1, identifier-3, identifier-4, identifier-6 and identifier-8 must be DB-KEY items.

2. Identifier-2 must be an alphanumeric elementary item. It must be large enough to handle any REALM-NAME. The minimum size is 30 (PIC x(30)).

3. Identifier-5 must be an elementary integer item with a range at least equal to the range of a COMP-1 item.

4. Identifier-7 must be an elementary integer item with a range at least equal to the range of a COMP-2 item.

5. Each record-name, set-name, realm-name or key-name specified must be included in the subschema.

4.1.4 General Rules

Format 1

1. If record-name, set-name, realm-name or key-name is specified, the data-base-key value for the current record of record-name, set-name, realm-name or key-name is placed in the data item referenced by identifier-1.

2. If a record-name, set-name, realm-name or key-name is not specified, the data-base-key value for the current record of the run-unit is placed in the data item referenced by identifier-1.

3. The currency indicator involved can be neither NULL nor virtual.

Format 2

1. The realm-name that is derived from the data-base-key value in the data item referenced by identifier-3 or from the specified currency indicator is placed in the data item referenced by identifier-2, according to the rules for an alphanumeric elementary move.

2. If identifier-3 or a record-name, set-name or key-name is not specified, the name of the realm that is associated with the current record of the run-unit is placed in the data item referenced by identifier-2, according to the rules for an alphanumeric elementary move.

3. If a currency indicator is involved, it can be neither NULL nor virtual.

Format 3

1. The currency indicator of set-name must point to a record (owner or member) or indicate a position (virtual).

2. The data-base-key of the NEXT, PRIOR or OWNER record within set-name of this record or position is placed in identifier-4.

   The data-base-key of the NEXT or PRIOR record points to a position, or to a record included in the subschema. The DBCS may have to scan the database to perform the ACCEPT statement, and data-base-exceptions such as *REALM not ready* or *unavailable record required by DBCS* may occur.

   The NEXT or PRIOR record may be the owner itself (there is no "end-of-set" condition). The set occurrence may be empty.

Format 4

1. The number of lines per page of realm-name is moved into the user defined item identifier-5.

Format 5

1. The lowest (first) data-base-key value of realm-name is placed in identifier-6. If the optional record-name is present in the clause, the value supplied is the lowest (first) data-base-key in the realm for the range of this record-type.

Format 6

1. The number of pages of realm-name is placed in identifier-7. If the optional record-name is present, the size (in pages) of the record range within realm-name is placed in identifier-7.

Format 7

1. The currency indicator of key-name must point to a record or indicate a position (virtual).

2. The data-base-key of the NEXT or PRIOR record within key-name of this record or position is placed in identifier-8.

   The data-base-key of the NEXT or PRIOR record points to a position or to a record included in the subschema. The DBCS may have to scan the database to perform the ACCEPT statement, and data-base-exceptions such as *INDEX resource not available* or *unavailable record required by DBCS* may occur.

4.1.5 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	None	0000000	-	-	-	-
	Data base not open	0109100				
ACCEPT ... CURRENCY .	CRU null	0103200				
	Current-of-realm null or virtual	0103100	x			
	Current-of-record-type null	0103100		x		
	Current-of set type null or virtual	0103100			x	
	Current-of-key-type null or virtual	0103100				x
ACCEPT ... REALM-NAME	CRU null	0103200				
	Current-of-record-type null	0103100		x		
	Current-of-set-type null or virtual	0103100				
	Data-base-key inconsistent	0104100				
	Current-of-key-type null or virtual	0103100				x
ACCEPT ... NEXT/ PRIOR/ OWNER	Current-of-set-type null	0103100			x	
	Unavailable record required by DBCS	0102200	x		x	
	Realm not ready	0109100	x		x	
	Current-of-key-type	0103100			x	
	Record not found	0102100	x			
	Index resource not available	0170200			x	
	Index inconsistency	0173615	x			x

4.2 CONNECT

4.2.1 Function

This statement causes a record stored in the database to become an actual member of a set, in which the record has been declared with one of the following types of insertion-retention membership:

1. MANUAL OPTIONAL,
2. AUTOMATIC OPTIONAL
3. MANUAL MANDATORY
4. MANUAL FIXED

The current record of the run-unit is the object of the statement.

4.2.2 General Format

```
CONNECT [record-name] TO set-name

   [                          {  SETS    } ]
   [ RETAINING CURRENCY FOR { } ] .
   [                          { set-name } ]
```

4.2.3 Syntax Rules

1. Record-name is not required. If specified, record-name must be the name of the current record of the run-unit and must be declared in the subschema.
2. The current record of the run-unit must be defined to be a member of the set-type specified in the CONNECT statement with one of the following types of membership:

   a) MANUAL OPTIONAL

   b) AUTOMATIC OPTIONAL

   c) MANUAL MANDATORY

   d) MANUAL FIXED.

3. Set-name must be declared in the subschema.

4.2.4 General Rules

1. Execution of the CONNECT statement causes the current record of the run-unit to become an actual member of the specified set-type provided that it is not currently connected to an occurrence of this set-type.

2. The set occurrence to which the current record of the run-unit is to be connected is determined by the set selection criteria of the subschema. The record is connected to the set occurrence in accordance with the set ordering criteria of the set-type. Therefore, the subschema must include all data items and all set-types required by the set selection criteria of this record-type within this set-type and also must include all data-items required by the set ordering criteria of this set-type (sort-key-fields and no-duplicate-control-fields).

3. The realms in which the current record of the run-unit is stored and in which the records of the affected set are stored must be open in update.

4. If the RETAINING phrase is not specified, the current record of the run-unit becomes the current record of the set-type to which the record has been connected. Other currency indicators are not affected.

5. If the RETAINING phrase is specified, no currency indicator is affected.

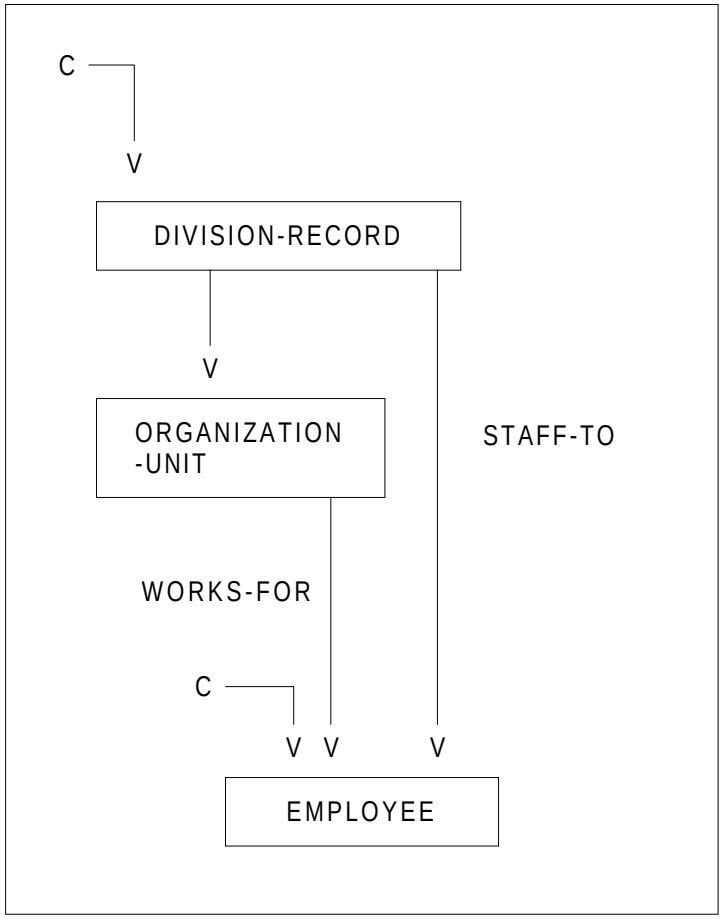
NOTES:

1. The set selection process uses the field or db-parameter values in the UWA. The set insertion process uses the sort-key and no-duplicate-control-field values of the record in the database.

2. If a data-base-exception occurs during the execution of a CONNECT statement, the database is restored to the state it was in before the statement was executed. This is performed without the use of journals. No currency indicators are affected by an unsuccessful statement.

Example:

The EMPLOYEE record is a MANUAL OPTIONAL, AUTOMATIC OPTIONAL, MANUAL MANDATORY or MANUAL FIXED member of the STAFF-TO set, with a set selection path consisting of only one level identified by CALC-KEY.



The DML sequence connecting an EMPLOYEE to a DIVISION-RECORD is given below:

```
     MOVE "mmm" TO EMPLOYEE-NUMBER.
FIND ANY EMPLOYEE.
     Establish current-of-run-unit
MOVE "nnnn" TO DIVISION-NUMBER.
     Prepare set selection path
CONNECT EMPLOYEE TO STAFF-TO.
     Connect
```

4.2.5 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	None; record connected	0000000	-	-	-	-
	Data base not open	0209100				
	CRU null	0203200				
	CRU not of correct type (when specified) or insertion/retention not compatible	0203300	x	x		
	CRU is a member already connected	0208100			x	
	The subschema does not include a data item or set type required by set selection or set ordering criteria	0204200		x		x
	Unvailable record required by DBCS	0202200				
	Realm not in ready state	0209100	x	x	(x)	
	Realm not in update mode	0209200	x	x	(x)	
	Access restriction is specified	0290100		x	x	
	Owner of new set occurrence does not respect the "AREA OF OWNER" condition	0273650	x	x	x	
	Set selection failed	0202300		x	x	(x)
	Duplicate not allowed (sort-key or other control field)	0205100		x	x	
	Database/Index inconsistency	0273615	x			x

4.3 DISCONNECT

4.3.1 Function

This statement logically removes a record from a specified set if the record is an OPTIONAL member of the set. The current record of the run-unit is the object of the statement.

4.3.2 General Format

```
DISCONNECT [record-name] FROM set-name.
```

4.3.3 Syntax Rules

1. Record-name is not required. If specified, record-name must be the name of the current record of the run-unit and must appear in the subschema.

2. The current record of the run-unit must be defined as an OPTIONAL member of the set-type specified in the DISCONNECT statement.

3. Set-name must be declared in the subschema.

4.3.4 General Rules

1. Execution of the DISCONNECT statement causes the current record of the run-unit to be removed from membership in the specified set-type provided that it is currently connected to an occurrence of this set-type.

2. The realms in which the current record of the run-unit is stored and in which the records of the affected set are stored must be open in update.

3. If the current record of the run-unit is the current of the set-type specified, the currency indicator for the set-type is updated to identify the position between the two records that the disconnected record occupied. If the current record of the run-unit is not the current record of the set-type specified, the currency indicator is not affected. Other currency indicators are not affected.

NOTES:

1. The current record of the specified set-type may become unavailable, as described in General Rule 3. However, the NEXT, PRIOR, and OWNER records remain available. This means that the currency indicator is valid for all subsequent DML statements which use a set position rather than a particular record. For example, the following sequence is valid:

   ```
   DISCONNECT FROM set-name_1.

   FIND NEXT WITHIN set-name_1.
   ```

2. If a data-base-exception occurs during the execution of a DISCONNECT statement, the database is restored to the state it was in before the statement was executed (without using journals). In addition, no currency indicators are affected by an unsuccessful statement.

4.3.5 DB-STATUS Values

FUNCTION CATEGORY	**ERROR CONDITIONS**	**DB-REGISTERS**				
		DB-STATUS	**REALM**	**RECORD**	**SET**	**KEY**
	None record disconnected	0000000	-	-	-	-
	Data base not open	0309100				
	CRU null	0303200				
	CRU not of correct type (when specified) or not declared as OPTIONAL member of the set	0303300		X (cru)	(X)	
	CRU is an OPTIONAL member of the set but currently not connected	0308300		X	X	
	Unavailable record required by DBCS	0302200				
	Realm not in ready state	0309100	X			
	Realm not in update mode	0309200	X			
	Access restriction is specified	0390100		(X)	(X)	

4.4 ERASE

4.4.1 Function

Removes one or more records from the database. The current record of the run-unit is the object of the statement.

4.4.2 General Format

```
                        [{SELECTIVE}          ]
ERASE [record-name]     [{PERMANENT} MEMBERS  ] .
                        [{ALL      }          ]
```

4.4.3 Syntax Rules

Record-name is not required. If specified, record-name must be the name of the current record of the run-unit and must be included in the subschema.

4.4.4 General Rules

1. Execution of the ERASE statement causes one or more records to be removed from the database.

2. If ALL, PERMANENT or SELECTIVE is not specified and the current record of the run-unit is not the owner of a set that currently has members, the record is disconnected from the sets to which it is currently connected, if any. The record KEY Entries, if any, are removed for all of the record keys defined in the record and then the record is removed from the database.

3. If ALL, PERMANENT or SELECTIVE is not specified and the current record of the run-unit is the owner of a set that currently has members, an exception condition results if the member records have OPTIONAL or MANDATORY membership. If the member records have FIXED membership, then the removal of the current record of the run-unit is executed as explained in General Rule 2 and is followed by the removal of all member records. Any record so removed is treated as though it were the object of an ERASE statement which did not specify ALL, PERMANENT or SELECTIVE.

   The process is repeated until all hierarchically related records have been processed.

4. If ALL is specified, and the current record of the run-unit is the owner of a set that currently has members, then the removal of the current record of the run-unit is executed as explained in General Rule 2, above, and is followed by the removal of all of member records. Any record so removed is treated as though it were the object record of an ERASE...ALL statement.

   The process is repeated until all hierarchically related records have been removed from the database.

5. If PERMANENT is specified, and the current record of the run-unit is the owner of a set that currently has members, then the removal of the current record of the run-unit is executed as explained in General Rule 2, above, and is followed by the removal of all MANDATORY or FIXED member records and the disconnection from the set to which the member records are currently connected for all OPTIONAL member records. Any record so removed is treated as though it were the object record of an ERASE...PERMANENT statement.

   The process is repeated until all hierarchically related records have been processed.

6. If SELECTIVE is specified, and the current record of the run-unit is the owner of a set that currently has members, then the removal of the current record of the run-unit is executed as explained in General Rule 2, above, and is followed by the removal of all MANDATORY or FIXED member records. These OPTIONAL member records are removed only if they do not currently participate as members in other set occurrences; otherwise they are only disconnected from the set to which they are currently connected. Any record so removed is treated as though it were the object record of an ERASE...SELECTIVE statement.

   The process is repeated until all hierarchically related records have been processed.

NOTE: For the reasons mentioned in the general rules above, the ERASE statement must always be used with care.

7. If ALL, SELECTIVE or PERMANENT is specified, and the current record of the run-unit is not the owner of a set that currently has members, execution proceeds as if ALL, SELECTIVE or PERMANENT had not been specified.

8. For an ERASE ALL, PERMANENT or SELECTIVE statement, the subschema must include all the set-types and record-types impacted in the hierarchy.

9. The realms in which the current record of the run-unit is stored and in which the records of the affected sets are stored must be opened in update.

10. Currency indicators are affected as follows:

    a) The current record of the run-unit is nulled.
    b) If any record removed from the database is the current record of its record-type, the currency indicator for the record-type is nulled.
    c) If any record removed from the database is the current record of a set-type of which it is the owner, the currency indicator for the set-type is nulled. If any record disconnected from a set to which it is currently connected is the current record of this set-type of which it is owner, the currency indicator for this set-type is unchanged.
    d) If any record removed from the database or disconnected from a set is the current record of the set-type to which it is connected, the currency indicator for the set-type is updated to identify the position between the two records that the removed record occupied.
    e) If any record removed from the database is the current record of its realm, the currency indicator for the realm is updated to identify the position that the removed record occupied.
    f) If any record removed from the database is the current record of a key-type, the currency indicator for the key-type is updated to identify the position that the removed record occupied.

NOTES:

1. As specified in General Rules 2 to 7, above, each object record is disconnected from any sets to which it is currently connected. This means no special consideration need be made for OPTIONAL sets. If an object record is currently connected to an OPTIONAL set, it will be disconnected and then eventually removed from the database, depending on the ERASE statement. If an object record is currently disconnected from an OPTIONAL set, no error will result and the record will be removed from the database or the record will remain unaffected, depending on the ERASE statement.

2. The current record of a set-type may become unavailable as described in General Rule 10d, above; however, the NEXT, PRIOR, and OWNER records remain available. This means that the currency indicator is valid for subsequent DML statements referencing a position within the set rather than a particular record of the set. For example, the following sequence is valid:

   ```
   ERASE B.

   FIND NEXT WITHIN A-B.
   ```

   where record B is a member of set A-B.

3. The current record of a realm may become unavailable as described in General Rule 10e; however, the NEXT and PRIOR records remain available. This means that the currency indicator is valid for subsequent DML statements referencing a position within the realm rather than a particular record of the realm.

4. The current record of a key-type may become unavailable as described in General Rule 10f; however, the NEXT and PRIOR records remain available. This means that the currency indicator is valid for subsequent DML statements referencing a position within the key rather than a particular record.

5. a) If a data-base-exception occurs during the execution of an ERASE statement without member deletion, the database is restored to the state it was in before the statement was executed (without using journals). The currency indicators are not affected and control is given back to the calling program with the appropriate DB-STATUS value.

   b) If a data-base-execption occurs during the execution of an ERASE statement with member deletion, but before the updating of the database has actually begun, then the DBCS proceeds as described in Note 5a, above.

   c) If a data-base-exception occurs during the execution of an ERASE statement with member deletion, but after the updating of the database has actually begun (that is, the ERASE statement is directed to an area which is neither in the ready state nor in update mode), then the processing depends on the protection level on the database as described below:

   - If the database is protected by the Before Journal in a BATCH, IOF or TDS environment, then it is restored to the state that it was in before the statement. In addition, currency indicators are not affected and control returns to the calling program with the appropriate DB-STATUS value.

   - If the database is not protected by the Before Journal in BATCH or IOF environment, it is left in an inconsistent state and the step is aborted. The database must be restored by FILREST or VOLREST.

   - In the TDS environment, if the database is protected by the After Journal and by deferred updates, it is restored to the state it was in at the last commitment point. The TPR is aborted and not restarted.

6. If neither ALL, PERMANENT nor SELECTIVE is specified and if a record can be accessed by either of the following access paths:

   - by a FIXED set

   - by a MANDATORY or OPTIONAL set

   The result of the statement will depend on which access path is taken to access the record the first time, as described below:

   - if the first access path was by a FIXED set, the member record is deleted,

   - if the first access path was by a MANDATORY or OPTIONAL set, the exception code (07200) is returned to the user.

4.4.5 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
General	None; record(s) erased	0000000	-	-	-	-
	Data base not open	0409100				
	CRU null	0403200				
	CRU not of correct type, when specified	0403300		x (cru)		
	Realm not in ready state	0409100	x	x		
	Realm not in update mode	0409200	x	x		
	Index inconsistency	0473615	x			x
	The sub-schema doesn't include a data item or set type required by set selection or set ordering criteria	0404200				
	Index resource not available	0470200	x	x		x
	An unavailable record is required by the DBCS	0402200				
	Acces restriction is specified	0490100		x		
ERASE [record name]	Record owner of a non empty set and record has optional or mandatory members	0407200	x	x	x	

4.5 FIND

4.5.1 Function

This statement establishes a specific record occurrence in the database as the object of subsequent statements.

4.5.2 General Format

```
FIND record-selection-expression

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

4.5.3 Syntax Rules

Each record-name, set-name or key-name specified must be included in the subschema.

4.5.4 General Rules

1. The FIND statement causes the record referenced by the record-selection-expression to become the current record of the run-unit. The realm-name for the record found is placed in the special register DB-REALM-NAME and the name of the record found is placed in DB-RECORD-NAME.

2. The DBCS only searchs in the database within the records included in the subschema.

3. If the RETAINING phrase is not specified, the record referenced by the record-selection-expression becomes the current record of its realm, the current record of its record-type, the current record of all set-types in which it is a tenant, and the current record of all key-types which are declared for this record.

4. If the RETAINING phrase with the optional word REALM is specified, the realm currency is not changed.

5. If the RETAINING phrase with the optional word RECORD is specified, the record-type currency indicator is not changed.

6. If the RETAINING phrase with the optional word SETS is specified, no set-type currency indicators are changed.

7. If the RETAINING phrase with the optional set-name is specified, the set currency indicators for the named set-types are not changed.

8. If the RETAINING phrase with the optional word KEYS is specified, no key-type currency indicators are changed.

9. If the RETAINING phrase with the optional key-name is specified, the key currency indicators for the named key-types are not changed.

10. If the RETAINING phrase with the optional word MULTIPLE is specified, then the realm, record-type, set-type and key-type currency indicators are not changed.

NOTES:

1. The execution of a FIND statement does not make the selected record contents available to the program; it merely identifies the record for use in subsequent statements. To obtain the contents of data items within the record, a GET statement must be issued.

2. If an exception condition is encountered, no currency indicators are changed. However, the special registers DB-REALM-NAME, DB-RECORD-NAME, DB-KEY-NAME, and DB-SET-NAME are updated.

3. The currency indicators for a set will not be updated if the record found is not currently connected to the set. A record may not be connected (to the set) if it is a MANUAL FIXED, MANUAL MANDATORY, MANUAL OPTIONAL or AUTOMATIC OPTIONAL member of the set. The currency indicators will be updated if the record found is currently connected to the set.

4.5.5 Format 1

```
FIND [ record-name-1 ] DB-KEY is identifier-1.
```

4.5.5.1 Syntax Rules

1. Identifier-1 must be defined as a DB-KEY item.

2. Record-name-1, if specified, must be included in the subschema.

4.5.5.2 General Rules

1. The record identified is the record whose data-base-key value is equal to the value of the data item referenced by identifier-1.

2. Record-name-1 is not required. If specified, the record found must be of the record-type specified by record-name-1.

3. The value in identifier-1 must be a complete data-base-key and not a realm-key. ACCEPT statements can be used to obtain data-base-key values.

4.5.6 Format 2

```
     {   ANY   }
FIND {         } record-name-2 .
     {DUPLICATE}
```

4.5.6.1 Syntax Rules

1. Record-name-2 must be included in the subschema.

2. The record referenced by record-name-2 must have a CALC location mode.

3. All the CALC-KEY items must be included in the subschema.

4.5.6.2 General Rules

1. With FIND ANY, a realm is selected, if there is a choice. The AREA IDENTIFICATION parameter is used (WITHIN clause in DDL).

2. With FIND ANY, the CALC-KEY value for the record named by record-name-2 is used to calculate the CALC-chain identification for the identified record. The value must be moved into the CALC-KEY fields in the UWA before the FIND statement is issued.

3. With FIND DUPLICATE, the DBCS searches within the CALC-chain, beginning the search with the current record of the run-unit, and looks for the next record of the same type whose CALC-KEY is equal to that of the current of run-unit. The FIND DUPLICATE statement assumes that the current of run-unit is already one of the duplicates.

NOTE: The value of the CALC-KEY used during the execution of a FIND DUPLICATE statement is the one found in the record in the database. The value in the UWA is not used during this execution.

4.5.7 Format 3

```
      {   ANY   }
FIND  {         } [record-name-2] USING key-name-1 .
      {DUPLICATE}
```

4.5.7.1 Syntax Rules

1. Record-name-3, if specified, must be included in the subschema.

2. Key-name-1 must be included in the subschema as a key of record-name-3, if specified.

4.5.7.2 General Rules

1. With FIND ANY, the record identified is the record whose key value is equal to the value of appropriate fields in the UWA. If the key is mono-record-type within the subschema, appropriate fields are the fields of the key in the record specified. If the key is multi-record-type within the subschema and if record-name-3 is not specified, appropriate fields are the key db-parameters. The key value must be moved into the suitable fields before the FIND statement is issued.

2. With FIND DUPLICATE, the record identified is the next record whose key value is equal to that of the current of the key referenced by key-name-1. If record-name is specified, the record identified is the next record of the record-type specified whose key value is equal to that of the current of the key referenced by key-name-1.

3. If record-name is not specified, all record-types of key-name-1 are considered. If record-name-3 is specified, only the record-type of record-name-3 will be considered.

NOTE: The key value used during the execution of a FIND DUPLICATE statement is the one found attached to the record in the database. The value in the UWA is not used during this execution.

4.5.8 Format 4

```
FIND [ record-name-4 ] FROM key-name-2.
```

4.5.8.1 Syntax Rules

1. Record-name-4, if specified, must be included in the subschema.

2. Key-name-2 must be included in the subschema as a key of record-name-4, if specified.

4.5.8.2 General Rules

1. The record identified is the record whose key value is equal to or greater than the value of appropriate fields in the UWA, according to the key order. If the key is mono-record-type within the subschema, appropriate fields are the fields of the key in the record. If the key is multi-record-type within the subschema, appropriate fields are the key db-parameters. The key value must be moved into the appropriate fields before the FIND statement is issued.

2. If record-name is not specified, all record-types of key-name-2 are considered. If record-name-4 is specified, only the record-type of record-name-4 will be considered.

4.5.9 Format 5

```
FIND DUPLICATE WITHIN set-name-1 USING { identifier-2 }... .
```

4.5.9.1 Syntax Rules

1. Set-name-1 must be included in the subschema.

2. Identifier-2 must be defined in the RECORD Entry for the current record of the set-type referenced by set-name-1.

4.5.9.2 General Rules

1. The record identified by the FIND DUPLICATE WITHIN statement:

   a) is a member of the set occurrence identified by the currency indicator for set-name-1.
   b) has a record-type equal to the current record of the set-type referenced.
   c) contains the contents of the data items referenced by identifier-2 equal to those in the current record of the set-type referenced.
   d) has a length equal to that of the current record of the set-type referenced if the record is of varying length, even if the DEPENDING ON control field is not specified in the USING clause.

2. The DBCS begins its search at the next record in the forward direction.

NOTES:

1 • The values of identifier-2 used in the search are those in the database record itself. The values in the UWA are not used during the execution of the FIND DUPLICATE statement.

2 • If the owner record is encountered before a duplicate record is found, a *record-not-found* (02400) exception condition results. If the duplicate record exists between the first and the current record of the set, it will not be found.

4.5.10 Format 6

```
     {NEXT        }
     {PRIOR       }                          {set-name-2  }
FIND {FIRST       } [record-name-5] WITHIN {realm-name-1} .
     {LAST        }                          {key-name-3  }
     {integer-1   }
     {identifier-4}
```

4.5.10.1 Syntax Rules

1. Integer-1 may be signed. This allows a relative forward or backward reference.

2. The data item referenced by identifier-4 must be a signed elementary integer.

3. Record-name-5, if specified, must be included in the subschema.

4. Set-name-2, realm-name-1 or key-name-3 must be included in the subschema.

4.5.10.2 General Rules

1. If record-name is not specified, all record-types declared in the subschema are considered in evaluation of the record-selection-expression.

2. If record-name is specified, only the record-type of the record referenced by record-name-5 is considered in evaluating the record-selection-expression. All other record-types are ignored.

3. If the NEXT phrase and realm-name are specified, the record identified is the one whose data-base-key is the next highest relative to the data-base-key value of the current record of the referenced realm.

4. If the PRIOR phrase and realm-name are specified, the record identified is the one whose data-base-key value is the next lowest relative to the data-base-key value of the current record of the referenced realm.

5. If the FIRST phrase and realm-name are specified, the record identified is the one whose data-base-key value is the lowest relative to all records stored in the referenced realm.

6. If the LAST phrase and realm-name are specified, the record identified is the one whose data-base-key value is the highest relative to all records stored in the referenced realm.

7. If integer-1 or identifier-4 is specified, and realm-name is specified, the record identified is the one whose ordinal position in the realm is equal to the value of integer-1 or to the contents of the data item referenced by identifier-4.

   If the specified value is positive, the ordinal position is relative to the record whose data-base-key value is the lowest in the referenced realm. If the specified value is negative, the ordinal position is relative to the record whose data-base-key value is the highest in the referenced realm. The content of the data item referenced by identifier-4 must have a non-null value.

8. If the NEXT phrase and set-name are specified, the record identified is the next record in the set relative to the current record of the set, according to the conventional set order.

9. If the PRIOR phrase and set-name are specified, the record identified is the prior record in the set relative to the current record of the set, according to the conventional set order.

10. If the FIRST phrase and set-name are specified, the record identified is the first member of the set in which the current record of the set is a member, according to the conventional set order.

11. If the LAST phrase and set-name are specified, the record identified is the last member of the set in which the current record of set is a member, according to the conventionnal set order.

12. If integer-1 or identifier-4 is specified, and set-name is specified, the record identified is the one whose ordinal position (in the set in which the current record of the set is a member) is equal to the value of integer-1 or to the contents of the data item referenced by identifier-4. If the value specified is positive, the ordinal position is relative to the first member of the set, according to the conventionnal set order. If the value is negative, the ordinal position is relative to the last member of the set, according to the conventionnal set order. The contents of the data item referenced by identifier-4 must have a non-null value.

13. If the NEXT phrase and key-name-3 are specified, the record identified is the next record, relative to the current of key, according to the key order.

14. If the PRIOR phrase and key-name-3 are specified, the record identified is the prior record, relative to the current record of the key, according to the key order.

15. If the FIRST phrase and key-name-3 are specified, the record identified is the one whose key value is the lowest, relative to all records of key.

16. If the LAST phrase and key-name-3 are specified, the record identified is the one whose key value is the highest, relative to all records of key.

17. If integer-1 or identifier-4 is specified, and key-name is specified, the record identified is the one whose ordinal position in the key is equal to the value of integer-1 or to the contents of the data item referenced by identifier-4. If the value specified is positive, the ordinal position is relative to the record whose key value is the lowest in the referenced key. If the value is negative, the ordinal position is relative to the record whose key value is the highest in the referenced key. The content of the data item referenced by identifier-4 must have a non-null value.

4.5.11 Format 7

```
                                  [       { realm-name-2 } ]
FIND CURRENT [ record-name-6 ] [WITHIN { set-name-3   } ]
                                  [       { key-name-4   } ]
```

4.5.11.1 Syntax Rules

1. Record-name-6, realm-name-2, set-name-3 or key-name-4, if specified, must included in the subschema.

2. If record-name-6 and set-name-3 are specified, record-name-6 must be included in the subschema as a member of set-name-3.

3. If record-name-6 and key-name-4 are specified, key-name-4 must be a key for record-name-6.

4.5.11.2 General Rules

1. If set-name-3 is specified, the record identified is the current record of the set-type indicated.

2. If realm-name-2 is specified, the record identified is the current record of the realm indicated.

3. If key-name-4 is specified, the record identified is the current record of the key indicated.

4. If record-name-6 and set-name-3 are specified, then record-name-4 must be the current record of the set indicated.

5. If record-name-6 and realm-name-3 are specified, then record-name-6 must be the current record of the realm indicated.

6. If record-name-6 and key-name-4 are specified, then record-name-6 must be the current record of the key indicated.

7. If record-name-6 is specified and the WITHIN clause is omitted, the record identified is the current record of the record-type indicated.

8. If neither realm-name-2, set-name-3, key-name-4 nor record-name-6 is specified, the record identified by the statement is the current record of the run-unit.

4.5.12 Format 8

```
FIND OWNER WITHIN set-name-4.
```

4.5.12.1 Syntax Rules

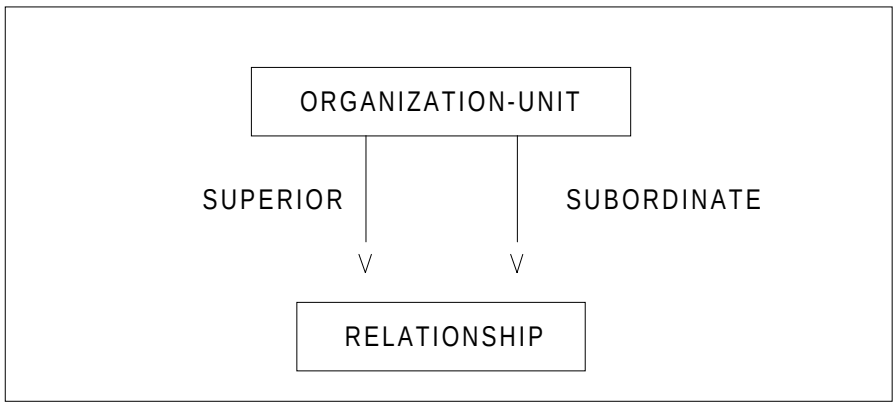
1. Set-name-4 must be included in the subschema.

4.5.12.2 General Rules

1. The record identified is the owner of the set occurrence identified by the currency indicator for set-name-4.

2. In some complex network structures, it is advisable to use the optional RETAINING CURRENCY FOR ... clause to insure that other set currencies are not altered by the execution of the FIND statement. When the owner record is found, it becomes the current record of all set-types in which it is tenant.

For example, a network structure for an organisation chart application can be illustrated as shown below:



When traversing the SUPERIOR set to locate all other ORGANIZATION-UNIT records which are subordinate to the current ORGANIZATION-UNIT record, the FIND OWNER WITHIN SUBORDINATE statement is issued after each RELATIONSHIP record has been found. The new ORGANIZATION-UNIT record will reset the current of set-type indicator for the SUPERIOR set, thereby terminating the initial retrieval. If, instead, the statement were written: FIND OWNER WITHIN SUBORDINATE RETAINING CURRENCY FOR SETS, then when the new owner ORGANIZATION-UNIT record is found, the current record of SUPERIOR set is unchanged, allowing the next RELATIONSHIP record to be found.

4.5.13 Format 9

```
FIND record-name-7 WITHIN set-name-5 [ CURRENT ]
                   [ USING { identifier-5 }... ].
```

4.5.13.1 Syntax Rules

1. Identifier-5, ... must be defined in the RECORD Entry for the record-type record-name-7, in the subschema.

2. Record-name-7 must be defined in the subschema as a member of the set-type set-name-5.

3. If the CURRENT phrase is not specified, the subschema must include all data items and set types required by the set selection criteria of the record within the set-type. If the set selection is dependent on any set-type currency indicator, this set-type must be included in the subschema.

4.5.13.2 General Rules

1. The record identified has a type equal to that of the record referenced by record-name-7.

2. If USING is specified, the record identified has the contents of the data items referenced by identifier-5 equal to those data items in the UWA.

   If the record is of varying length, the length of the record identified is equal to the length of the record in UWA, even if the DEPENDING ON control field is not specified in the USING clause.

   If USING is not specified, the record identified is the first record in the set occurrence.

3. The DBCS proceeds in its search as follows:

   a) If CURRENT is not specified, the correct set occurrence is selected according to the set selection rules.

   (i) The entry point record is selected by one of four methods: by APPLICATION (currency indicator for set-type); by CALC-KEY (randomize a specified control field); by KEY (key fields supplied by program) or by DATA-BASE-KEY (data-base-key of record supplied by program).
   (ii) Each intervening set is searched based on the keys defined in the Schema DDL. Each set searched corresponds to a THEN THRU clause in the set selection clause.
   (iii) The last set searched identifies the set occurrence in which the identified record is a member. If there are no THEN THRU clauses in the set selection clause, the entry point record identifies the set occurrence in which the identified record is a member.

b) If CURRENT is specified, the set occurrence is identified by the current of set-type.

c) The set occurrence, identified in the manner described above, is searched in the forward direction to locate (if USING is specified) the record in which the contents of identifier-5 are equal to the contents of those data items in the UWA. If USING is not specified, the first record is selected. Only records of the type referenced by record-name-7 are considered in this search. Other record-types are ignored.

NOTES:

1. If the entry point record as described in General Rule 3a(i) is identified by CALC-KEY and the record can reside in more than one realm, realm selection will be invoked by using the AREA IDENTIFICATION parameter supplied by the schema. The CALC-KEY for this record must be initialized in the UWA before executing the FIND statement.

2. If the entry point record as described in General Rule 3a(i) is identified by APPLICATION, the current record of the set-type specified in the set selection must be established before executing the FIND statement. This set-type is the one referenced by the first THRU phrase of the SELECTION clause of the MEMBER subentry. The record identified may be either an owner or member of the set-type.

3. If the entry point record as described in General Rule 3a(i) is identified by KEY, the key fields for this record must be initialized before executing the FIND statement.

4. If the entry point record as described in General Rule 3a(i) is identified by DATA-BASE-KEY, the data-base-parameter specified in the subschema must be set to the correct value before executing the FIND statement. The record identified must be the owner of the set.

5. The set selection key (if any) as described in General Rule 3a(ii) must be set to the correct values in the UWA before executing the FIND statement.

6. If no set occurrence is found which satisfies the set selection criteria as described in General Rule 3a, the exception condition *set selection not satisfied* (02300) is returned. This can occur during any of the steps described in General Rule 3a.

7. If the correct set occurrence is found, but a record cannot be found that satisfies the record-selection-expression as described in General Rule 3b, the exception condition *record-selection-expression not satisfied* (02400) is returned.

8. The search of the occurrence as described in General Rule 3b continues through the entire set occurrence, beginning at the first member of the set in the set occurrence.

4.5.14 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
General	None; record found	0000000	x	x	-	-
	Data base not open	0509100				
	Realm not in ready state	0509100	x	x		
	Index resource unavailable	0570200	x	(x)		x
	Database/Index inconsistency	0573615	x			x
	Access restriction is specified	0590100		(x)	(x)	
	Unavailable record required by the DBCS	0502200			(x)	(x)
FIND DB-KEY IS...	Data-base-key inconsistent or does not correspond to record-type range	0504100				
	Valid data-base-key but no record found or record found has not the correct type (when specified)	0502400	x	(x)		
FIND FROM key	No record found	0502400				
	Illegal data item	0505200				
	Realm name in area-id is unknown	0504300				

4.6 FINISH

4.6.1 Function

This statement ends the availability of one or more realms to the program.

4.6.2 General Format

```
FINISH [realm-name-1].
```

4.6.3 Syntax Rules

1. Realm-name-1 is not required. If specified, realm-name1 must be the name of a realm included in the subschema.

2. The same realm-name must not be specified more than once in a single FINISH statement.

4.6.4 General Rules

1. Execution of the FINISH statement ends the availability of the realms(s).

2. If realm-name-1 is specified, the ready state of the indicated realms is terminated.

3. If realm-name-1 is not specified, the ready state of all the realms included in the subschema is terminated.

4. At the completion of the execution of a FINISH statement, all currency indicators that identify subschema records stored in the affected realm(s) are made null.

5. All affected realms must be in ready state at the beginning of execution of a FINISH statement. If the ready mode of all object realms cannot be terminated, no ready mode is terminated, and the execution of the FINISH statement is unsuccessful.

4.6.5 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	None	0000000	-	-	-	-
	Data base not open	0609100				
	Realm not in ready state	0609100	X			

4.7 GET

4.7.1 Function

This statement makes some or all the fields in a database record available to the program. The object of the statement is the current record of the run-unit.

4.7.2 General Format

Format 1

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

Format 2

```
GET      {identifier-1}.
```

4.7.3 Syntax Rules

1. In Format 1, record-name, if specified, must appear in the subschema.

2. In Format 2, identifier-1 ... must reference data items, elementary or not, which are defined in the RECORD Entry of the subschema for the current record of the run-unit.

4.7.4 General Rules

1. Execution of a GET statement places all or part of the current record of the run-unit in the associated UWA record.

2. In Format 1, record-name, if specified, must be the name of the current record of the run-unit.

3. In Format 1, the contents of the current record are moved to the UWA. All other UWA fields will be unchanged.

4. In Format 2, only those data items listed will be moved to the UWA record associated with the current record of the run-unit. All other UWA fields will be left unchanged.

5. The data items are moved into the UWA in accordance with the data transFormation rules between data types. If a data transFormation rule is violated, a data-base-exception occurs and the GET statement is denied.

6. No currency indicator is affected by the GET statement.

4.7.5 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	None	0000000	-	-	-	-
	Data base not open	0809100				
	Area not in ready state	0809100	X			
	CRU null	0803200				
	CRU not of correct type (if specified)	0903300		X (cru)		
	Access restriction is specified	0890100		X		
	CRU UWA record-description missing	0873660		X		
	TransFormation of a data item violates a data transformation rule	0805300		X		

4.8 DATABASE CONDITION (IF)

4.8.1 Function

This statement enables the program to take decisions depending upon the value (or condition) which is obtained following a test directed to a current record and/or to its set membership. There are two database conditions:

1. **The Tenancy Condition,** which indicates the status of a record with respect to one or several set occurrences.

2. **The Membership Condition,** which indicates the status of a set occurrence.

These conditions are described in the General Rules in the following subsections.

4.8.2 General Format

For the Tenancy Condition

```
            {OWNER }
[set-name] {MEMBER}
            {TENANT}
```

For the Membership Condition:

```
set-name is [NOT] EMPTY
```

4.8.3 Syntax Rules

1. The Database Condition is a simple condition from a COBOL point of view; as such, it may appear anywhere where a simple condition may be used.

2. The Database Condition has a truth value of 'true' or 'false'. This condition cannot be tested when a data-base-exception occurs.

3. Set-name, if specified, must be included in the subschema.

4.8.4 General Rules

1. A set occurrence will be considered as empty if it contains none of the records included in the subschema. If this is the case, a data-base-exception, such as *realm not open* or *unavailable realm required by the DBCS* may occur.

2. Currencies are left unchanged after the evaluation of a Database Condition.

4.8.5 General Rules for the Tenancy Condition

1. The Tenancy Condition indicates whether or not the current record of the run-unit is presently defined as one of the following:

   - the owner of a non-empty set occurrence of a specified set-type (or at least one set-type)

   - or an actual member of a set occurrence of a specified set-type (or at least one set-type)

   - or both of the above.

2. The object set occurrence is selected through the current-of-run-unit. The OWNER condition will be evaluated by taking into account only the records declared in the subschema as members of the object set occurrence(s).

3. Set-name OWNER 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 as the owner type of this set-type, otherwise a data-base-exception occurs.

4. OWNER is true when the current-of-run-unit is the owner of a non-empty set occurrence of at least one set-type. The record-type of the current-of-run-unit must appear in the subschema as the owner type of at least one set-type, otherwise a data-base-exception occurs.

5. Set-name MEMBER 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 subschema as an OPTIONAL member of this set-type, otherwise a data-base-exception occurs.

6. MEMBER is true when the current-of-run-unit is an actual member of at least one set-type. The subschema set-types considered are those for which the record-type of the current-of-run-unit is declared as an OPTIONAL member. If no such set-type exists, a data-base-exception occurs.

7. Set-name TENANT is equivalent to:

   - Set-name OWNER, if the record-type of the current-of-run-unit is included in the subschema as owner-type of this set-type.

   - Set-name MEMBER, if the record-type of the current-of-run-unit is included in the subschema as an OPTIONAL member-type of this set-type.

If the record-type of the current-of-run-unit is declared neither as the owner-type nor as an OPTIONAL member-type of this set-type, a data-base-exception occurs.

8. TENANT is the logical "OR" of OWNER and MEMBER. The following elements must exist in the subschema:

   - either set-types for which the record-type of the current-of-run-unit is the owner-type
   - and/or set-types for which the record-type of the current-of-run-unit is an OPTIONAL member-type.

Otherwise a data-base-exception occurs.

NOTE: The Tenancy Condition cannot be used to interrogate the **type** data structure described in the subschema. It only deals with **occurrences** of the elements of the data structure. For instance, the purpose of IF OWNER is not to test whether the **record-type** of the current-of-run-unit appears in the subschema as the owner-type of some set-types; its purpose is to test whether the current-of-run-unit is currently the owner of at least one non-empty **set occurrence**.

4.8.6 General Rules for the Membership Condition

1. The Membership Condition indicates whether or not the occurrence of the specified set-type identified by its current of set-type currently has any members.

2. The current of set-type identifying the set occurrence must not be null or virtual (as a result of an ERASE statement), otherwise a data-base-exception occurs.

3. "Set-name IS EMPTY" is true if the set occurrence has no members.

4. When NOT is specified, the result of the Membership Condition test is reversed.

4.8.7 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	None; condition true	0000000	-	-	-	-
	None; condition false	0000000	-	-	-	-
	Data base not open	0909100	-	-	-	-
	Realm not open	0909100	x			
	Unavailable record required	0902200			x	
	Index resource not available	0970200			x	
Tenancy condition	CRU null	0903200				
	CRU type irrelevant to the condition (not owner or optional member)	0903300		x (cru)		
Membership condition	Current of set null	0903100			x	

4.9 MODIFY

4.9.1 Function

This statement alters the contents of one or more data items in a record and/or changes the sets membership of the record. The object of the statement is the current record of the run-unit.

4.9.2 General Format

Format 1

```
MODIFY { [record-name]    } [RETAINING-phrase]  .
       { {identifier-1}...}
```

Format 2

```
 MODIFY   [record-name]      ONLY { ALL            } MEMBERSHIP
                                  { {set-name-1}...}

                                  [ RETAINING-phrase] .
```

Format 3

```
 MODIFY {[record-name]  } INCLUDING { ALL          } MEMBERSHIP
        {identifier-1...}           {{set-name-1}..}

                               [ RETAINING-phrase] .

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

4.9.3 Syntax Rules

1. Record name and set names, if specified, must appear in the subschema.

2. Identifier-1 must reference data items included in the subschema.

3. Record-name, if specified, must be defined as a member of the set-types referenced by set-name (if any set-name references have been made).

4.9.4 General Rules (applicable to all Formats)

1. Record-name or identifier-1 must be of the same type as the current-of-run-unit. Otherwise a data-base-exception will occur.

2. If the RETAINING phrase is not specified, the object record becomes, on successful completion, the current record of its realm, the current record of its record-type, the current record of all set-types in which it is a tenant, and the current record of all the key-types in which it participates.

3. If the RETAINING phrase with the optional word REALM is specified, the realm currency indicator is not changed.

4. If the RETAINING phrase with the optional word RECORD is specified, the record-type currency indicator is not changed.

5. If the RETAINING phrase with the optional word SETS is specified, the set-type currency indicators are not changed.

6. If the RETAINING phrase with the optional set-name ... is specified , the set currency indicators for the named set-types are not changed.

7. If the RETAINING phrase with the optional word KEYS is specified, the key-type currency indicators are not changed.

8. If the RETAINING phrase with the optional key-name ... is specified, the key currency indicators for the named key-types are not changed.

9. If the RETAINING phrase with the optional word MULTIPLE is specified, then the realm, record-type, set-type and key-type currency indicators are not changed.

10. The realm in which the current record of the run-unit is stored must be open in update. Records of sets in which membership is affected by the execution of the MODIFY statement must be stored in realms that are open in update.

11. Records of sets referenced by the set selection criteria must be stored in realms that are in ready state.

4.9.5 Format 1

1. This Format replaces the contents of one or more items in the current record of the run-unit.

2. If record-name is omitted, the DBCS replaces the contents of all the items of the current record of the run-unit with the contents of the corresponding items in the UWA.

3. If record-name is specified, the DBCS checks that the current record of the run-unit is of the appropriate record-type and performs the action described in General Rule 2.

4. If identifier-1 ... is specified, the DBCS replaces the contents of the data items specified with the contents of the corresponding items in the UWA.

5. If the data item which is modified contains a CHECK clause invoking other items of the record, these items must be included in the subschema. Otherwise, the modification is denied and a data-base-exception occurs.

   If the data items modified do not satisfy the validity checks defined in the DDL for this record-type, the modification is denied and a data-base-exception occurs.

6. If CALC-key items are modified and the MIGRATION ALLOWED clause is not specified in the DMCL for this record-type, the record keeps its data-base-key and is logically inserted in the CALC chain corresponding the new CALC-key value.

   If an overflow-link record (OWL) corresponding to new CALC chain has to be created in the page of the record and if there is not sufficient space, the modification is denied and a data-base-exception occurs.

7. If CALC-key items are modified and the MIGRATION IS ALLOWED clause is specified, the record is moved to the bucket corresponding to the new CALC-key value. The record cannot change areas as a result of a migration. Set pointers and currency indicators referencing the moved record are updated to reflect its new position.

8. If modified items are sort-key items in a set (including the DATA-BASE-KEY in case of migration), the set is reordered according to its set ordering criteria.

9. If modified items are items in a key of the record, this key must be available for the subschema; otherwise, the modification is denied and a data-base-exception occurs.

10. If modified items are items in a key of the record, the associated key and the currency indicator of the key-type are updated to reflect the new position.

11. If the data item modified is a control field of a set (sort-key, no-duplicate-control-field) in which it is member, this set must be included in the subschema.

    If the modified items do not satisfy the no-duplicate criteria (CALC-KEY, sort-key, no-duplicate-control-fields), the modification is denied and a data-base-exception occurs.

12. In the case of varying record length, the following rules apply:

    a) The modification of a field whose length changes may cause the contraction or expansion of the record on the page. In the case of expansion, if there is not sufficient space in the page of the record to accomodate the expanded record and if the MIGRATION IS ALLOWED clause is not specified, the modification is denied and a data-base-exception occurs. Otherwise the record is moved to the next available position in the realm. The record cannot change realms as a result of a migration.
    b) If the DEPENDING ON control field is not specified in the MODIFY {identifier-1...} statement, the length of the current-of-run-unit is not modified.

    c) If the DEPENDING ON control field is specified in the MODIFY {identifier-1...} statement and if at least one controlled field is not specified, there are two possible results:

    - in the case of record contraction, an illegal decimal data may result, leading to a data-base-exception

    - in the case of record expansion, the current-of-run-unit is filled with blank characters and an illegal decimal data may result, leading to a data-base-exception.

4.9.6 Format 2

1. Format 2 changes the set membership of the current record of the run-unit for specified set-types.

2. If record-name is specified, the DBCS checks that the current record of the run-unit is of this record-type.

3. If ALL is specified, the affected set-types are the set-types of the subschema in which the record is a currently connected MANDATORY or OPTIONAL member. A FIXED member is not affected.

4. If set-name ... is specified, the affected set-types are those listed. If the record is defined as an OPTIONAL member of one of these set-types, it must be currently connected to this set.

   If the record is defined as a FIXED member of one of these set-types, the modification is denied and a data-base-exception occurs.

5. For each affected set-type, all data items and set-types required by the associated set insertion or set selection must be included in the subschema. Otherwise, the modification is denied and a data-base-exception occurs.

6. For each affected set-type, the set membership is changed as follows:

   a) The current record of the run-unit is disconnected from the set occurrence to which it belongs.

   b) A new occurrence is selected according to the set selection criteria specified either in the subschema (if any), or in the Schema DDL.

c) The current record of the run-unit is connected to the occurrence selected above. This is done according to the set ordering criteria specified in the Schema DDL.

7. If the current record of the run-unit has a LOCATION mode defined as "VIA set-name WITHIN AREA OF OWNER", a modification of the membership in this set will be denied if the new owner is not in the same realm as the current record of the run - unit.

8. If data items in the current record of the run-unit do not satisfy the no-duplicate criteria (sort-key, no-duplicate control-fields), a data-base-exception occurs.

4.9.7 Format 3

1. Format 3 replaces the contents of one or more items in the current record of run-unit and changes its set membership in one or more sets.

2. The actions performed by the DBCS are the combination of those described for Format 1 and Format 2.

NOTES:

1. OPTIONAL set membership can be changed with a MODIFY statement. If the current record of the run-unit is currently connected to the set, it is treated exactly as an MANDATORY MEMBER set. It is not possible to obtain the effect of a CONNECT statement with a MODIFY statement. If the record is connected to a set before the MODIFY statement, it will be connected afterwards. If the record is disconnected from a set before the MODIFY statement, it will be disconnected afterwards. For a FIXED set, the membership cannot be changed by MODIFY statement.

2. If one of several control-fields is to be changed, the one specified will be obtained from the UWA. The others required to execute the function will be obtained from the current record of the run-unit. This means it is possible to change any combination of data items in a record.

3. a) If a data-base-exception occurs during the execution of a MODIFY statement without migration, the database is restored to the state that it was in before the statement was executed (without the use of journals). The currency indicators are not affected and control returns to the calling program with the appropriate DB-STATUS value.

   b) If a data-base-exception occurs during the execution of a MODIFY statement with migration, but before the updating of the database has actually begun, the DBCS proceeds as described above.

c) If a data-base-exception occurs during the execution of a MODIFY statement with migration, but after the updating of the database has actually begun, the processing depends on the protection level on the database as described in the paragraphs below:

- In a BATCH, IOF or TDS environment, if the database is protected by the Before Journal, it is restored to the state that it was in before the statement was executed and control returns to the calling program with the appropriate DB-STATUS value.
- In a BATCH or IOF environment, if the database is not protected by the Before Journal, it remains in an inconsistent state and the step is aborted. In this case, the database must be restored by FILREST or VOLREST.
- In a TDS environment, if the database is protected by the After Journal and by deferred updates, it is restored to the state that it was in at the last commitment point; the TPR is aborted and not restarted.

4.9.8 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	None	0000000	-	-	-	-
	Data base not open	1109100				
	CRU null	1103200				
	CRU not of correct type or not member of the specified set	1103300		x		
	Access restriction is specified	1190100	x	x	x	
	Realm or data base not in ready state	1109100	x	x	(x)	
	Area not in update	1109200	x	x	x	
	An unavailable record is required by DBCS	1102200				
	Contents of data item are duplicated in the data base	1105100	(x)	x	(x)	
	Index resource is not available	1170200	x	x		x
	The subschema does not include a data or set-type required by set selection or set ordering	1104200			x	
	Database/Index inconsistency	1173615	x			x

DB-STATUS Values (continued)

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
Data modification	UWA record-description missing	1173660		x		
	Data items do not satisfy the validity checks	1105200		x		
	Transformation of a data item violates a data transformation rule	1105300		x		
	CALC-KEY /variable length record modification cannot be performed without migration	1173640		x		
	Space in realm is exhausted	1180200	x	x		
	Index space is exhausted	1180300				
Membership modification	Object record is not currently connected to the set	1108300		x	x	
	No set can be located to satisfy the set selection criteria	1102300			x	x
	Set membership violated AREA of OWNER clause	1173650	x (owner)	x	x	
	Set membership change is required	1108500		x	x	

4.10 READY

4.10.1 Function

This statement prepares one or more realms for processing.

4.10.2 General Format

```
      [                                {EXCLUSIVE  {RETRIEVAL}}]
      [                                {           {UPDATE   }}]
      [                                                        ]
READY [[realm-name-1]... USAGE-MODE IS{SHARED      RETRIEVAL }]
      [                                                        ]
      [                                {           {RETRIEVAL}}]
      [                                {{MONITORED}{UPDATE   }}]
```

4.10.3 Syntax Rules

1. Realm-name-1 is not required. If specified, realm-name-1 must be the name of a realm included in the subschema.

2. The same realm-name must not be specified more that once in a single READY statement.

4.10.4 General Rules

1. Execution of any READY statement affects an entire realm, regardless of the specific Format of the statement.

2. At the beginning of the execution of a READY statement, no realm affected by that statement may be in the ready state. If a realm is in the ready state at that time, the ready mode is not initiated and the execution of that statement is unsuccessful.

3. If realm-name-1... is specified, each realm referenced is prepared for access and placed in ready state.

4. If realm-name-1... is not specified, all realms included in the subschema are prepared for access and placed in ready state.

5. The USAGE-MODE phrase establishes the usage mode for the affected realms and thereby specifies the types of operation permitted. Usage mode remains in effect for the affected realms until a FINISH statement directed to those realms is executed, or until the termination of the run-unit which executed the READY statement.

6. The RETRIEVAL phrase permits access to read the contents of the affected realms.

7. The UPDATE phrase permits both access to the contents of the affected realms and their modification.

8. The EXCLUSIVE phrase specifies that the affected realms remain unaffected by concurrent run-units.

9. The SHARED phrase specifies that the affected realms can be read by concurrent run-units using the same usage-mode. Sharing is at file level in RETRIEVAL mode only.

10. The MONITORED phrase specifies that the affected realms can be read or modified by concurrent run-units using the MONITORED usage mode. Sharing is at page level (through the GAC facility).

11. The JCL ASSIGN/DEFINE statements must be consistent with the USAGE-MODE defined in the program: UPDATE will not be permitted with an ACCESS=READ parameter or a READLOCK=STAT parameter.

12. When USAGE-MODE is UPDATE, if there is no sharing with concurrent run-units and the Before Journal is not used, the affected realms are placed in the transient state. They remain in this state until the execution of a successful FINISH statement. If no successful FINISH statement is directed to a realm before the end of the current run-unit, that realm will remain in the transient state, whether the run-unit terminates successfully or not. The next time the realm is referenced by a READY statement, the job step will fail unless the user has requested that the transient state of that realm be ignored. (See Section 6 of this manual for further inFormation on Rrun-time commands.)

    If the transient state is ignored, the DBCS records the time and date of the last instance when the transient state was ignored. This is recorded in the IDS labels.

13. The execution of a READY statement does not affect the currency indicators.

14. READY statements coded in a TPR are ignored at run-time. Implicit READY statements with MONITORED usage-modes are generated by the DBCS for each transaction.

4.10.5 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	None	0000000	-	-	-	-
	Realm already in ready state	1309300	X			
	Access restriction (usage-mode not permitted)	1390100	X			
	Realm not assigned	1373590	X			
	EFN unknown	1373591	X			
	Usage-mode conflict with ASSING/DEFINE	1373592	X			

4.11 STORE

4.11.1 Function

This statement causes a record to be stored in the database and establishes the current record of the run-unit.

4.11.2 General Format

```
                      [                          {  MULTIPLE        }]
                      [                          {[ REALM           ]}]
                      [                          {[{SETS          }]}]
STORE record-name [ RETAINING CURRENCY FOR {[{{set-name} ...}]}] .
                      [                          {[ RECORD          ]}]
                      [                          {[{KEYS          }]}]
                      [                          {[{{key-name} ...}]}]
```

4.11.3 Syntax Rules

1. Each record-name, set-name or realm-name specified must be included in the subschema.

4.11.4 General Rules

1. The fields of the UWA record referenced by record-name are moved to the record in the database. This is accomplished in accordance with the data transFormation rule between data types. If a data transFormation rule is violated, a data-base-exception occurs and the STORE is denied. For all the fields of this record which are declared in a Schema DDL Data Subentry but which do not appear in the Subschema DDL Data Subentry, an initial value must have been specified. Otherwise, the STORE is denied and a data-base-exception occurs.

2. The record referenced by record-name becomes a member of each set-type for which it is an AUTOMATIC member. The sets to which the record is connected are determined by the set selection criteria. The record is connected in accordance with the set ordering criteria of each set-type.

   If a data item, record-type or set-type required by the set selection or set ordering criteria does not appear in the subschema, the STORE statement is denied and a data-base-exception occurs.

   If there is a set-type for which the record referenced by record-name is an AUTOMATIC member, this set must be included in the subschema. Otherwise, the STORE statement is denied and a data-base-exception occurs.

3. The referenced record is established as the owner of an empty set occurrence for each set-type of which it is an owner.

4. The STORE statement causes the referenced record to become the current record of the run-unit. The realm-name and record-name of the record are placed respectively in special registers DB-REALM-NAME and DB-RECORD-NAME.

5. If the RETAINING phrase is not specified, the referenced record becomes the current record of its realm, the current record of its record-type, the current record of all key-types declared on this record, and the current record of all set-types in which it has been declared to be an owner or AUTOMATIC member.

6. If the RETAINING phrase with the optional word REALM is specified, the realm currency indicator is not changed.

7. If the RETAINING phrase with the optional word RECORD is specified, the record-type currency indicator is not changed.

8. If the RETAINING phrase with the optional word SETS is specified, no set-type currency indicators are changed.

9. If the RETAINING phrase and *set-name* ... are specified, the currency indicators of *set-name* ... are not changed.

10. If the RETAINING phrase with the optional word KEYS is specified, no key-type currency indicators are changed.

11. If the RETAINING phrase and *key-name* ... are specified, the currency indicator of *key-name* ... are not changed.

12. If the RETAINING phrase with the optional word MULTIPLE is specified, the realm, record-type, set-type and key-type currencies are not changed.

13. The realm in which the record is stored must be open in update. Records of sets in which the stored record is an AUTOMATIC member must be stored in realms that are open in update.

14. Records of sets referenced by the set selection criteria must be stored in realms that are in ready state.

15. The record referenced by record-name remains available to the program in the UWA.

16. If the record has a VIA SET location mode, the corresponding set must be included in the subschema.

17. If the subschema specifies an access restriction that prohibits the STORE statement of the named record, the store is denied and a data-base-exception occurs.

4.11.5 Record Placement

The DBCS assigns a unique data-base-key to the record placed in the database as a result of the execution of the STORE statement. The schema definition, as well as the COBOL programmer, can specify certain constraints on the assignment of a data-base-key. These constraints depend on the location mode specified for the record and are listed below for the three possible location modes: DIRECT, CALC and VIA SET.

4.11.5.1 Constraints on a Record with DIRECT Location Mode

If the record has a DIRECT location mode, the contents of the specified data item are used to assign a data-base-key to the new record. The data item must contain a data-base-key or a realm-key. In the first case, the realm code is not involved in the realm selection and is ignored.

1. A realm is selected, if there is a choice, by means of the schema-specified AREA IDENTIFICATION parameter.

2. If the supplied realm-key is within the record range and is not assigned to an existing record, the DBCS will assign it to the new record.

3. If the supplied realm-key is within the record range and is already assigned to an existing record, the DBCS searches forward to find an available data-base-key. The first one found is assigned to the new record. If the end-of-range is encountered, the search is resumed at the beginning of the range.

4. If the supplied data-base-key is not within the record range, the DBCS returns an exception status.

NOTE: The DIRECT location mode should be avoided as much as possible since it does not lend itself to automatic reorganization by a standard utility.

4.11.5.2 Constraints on a Record with CALC Location Mode

If the record has a CALC location mode, the contents of the specified UWA data items are used to assign a data-base-key.

1. If a choice of realms exists, a unique realm is selected by means of the schema-specified AREA IDENTIFICATION parameter.

2. The location mode of a data item is used to determine the CALC bucket where the record is to be stored. The record is placed in the first page of the bucket or in a subsequent page.

4.11.5.3 Constraints on a Record with VIA SET Location Mode

If the record has a VIA SET location mode, the DBCS assigns a data-base-key based on the data-base-key of the owner record occurrence of the set-type specified for the location mode. The process of assigning a data-base-key and a physical location for a VIA SET record is described below:

1. The set selection criteria for the specified set is used to locate the owner record occurence of the set. This may involve searching a number of path levels before the required owner is found.

2. A realm is selected, if there is a choice, by means of the AREA IDENTIFICATION parameter specified in the schema.

3. If the owner record-type of the specified set and the new record-type are in different ranges of the same realm or in different realms, the new record is placed in its range proportional to the owner occurence in its range. **If both records are in the same range,** the new record is placed as near as possible to the owner occurence. If the set selection path consists of only one level identified by APPLICATION and if the set order is NEXT or PRIOR, the search for a free data-base-key starts from the position of the current-of-set and not from the position defined above. This optimizes processing time when the set occurence represents a volatile sequential file spreading over many pages.

NOTES:

1. Since the record stored is not connected to any MANUAL sets, the currency indicators will not be updated for MANUAL sets. This is true whether a RETAINING phrase is specified or not.

2. If the location mode is VIA SET and the record to be stored is a MANUAL member of the specified set, the new record is physically placed in the same way as if it were being established as a member of the set. Set selection is executed for the set to locate the correct owner occurence, but the new record is not connected to the set.

3. When a realm is selected for a record, the schema-specified parameter is used. The program executing the STORE statement must put the name of the correct realm in the parameter before executing the STORE statement.

4. Set selection rules for automatic sets and for the set specified in the VIA phrase may require that a realm be selected to locate the entry point record. This is necessary when the entry point record is a CALC record that is present in more than one realm. The COBOL program must move the correct realm-name to the AREA-ID parameter before executing the STORE statement.

5. Since there is no guarantee that a DIRECT record has been assigned the data-base-key which is supplied by the program, it is advisable to check whether it has been assigned (if this is relevant to your implementation). The data-base-key actually **assigned** can be obtained with an ACCEPT statement. This can be compared to the **supplied** data-base-key. The data item containing the data-base-key is not altered by the STORE statement, even if the record stored has not been assigned the data-base-key which as supplied by the program.

6. If a data-base-exception occurs during the execution of a STORE statement, the database is restored to the state it was in before the statement was executed (without the use of journals). In addition, currency indicators are not affected by an unsuccessful statement.

4.11.6 DB-STATUS Values

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
Data modification	UWA record-description missing	1173660		x		
	Data items do not satisfy the validity checks	1105200		x		
	Transformation of a data item violates a data transformation rule	1105300		x		
	CALC-KEY /variable length record modification cannot be performed without migration	1173640		x		
	Space in realm is exhausted	1180200	x	x		
	Index space is exhausted	1180300				
Membership modification	Object record is not currently connected to the set	1108300		x	x	
	No set can be located to satisfy the set selection criteria	1102300			x	x
	Set membership violated AREA of OWNER clause	1173650	x (owner)	x	x	
	Set membership change is required	1108500		x	x	

DB-STATUS Values (continued)

FUNCTION CATEGORY	ERROR CONDITIONS	DB-REGISTERS				
		DB-STATUS	REALM	RECORD	SET	KEY
	Duplicate item not allowed (CALC-key, sort-key, key, field)	1505100	(x)	x	(x)	(x)
	Set selection failed	1502300		x	x	(x)
	The subschema does not include a data item or set-type required by set selection or set ordering criteria	1504200		x	x	(x)
	Unavailable record required by DBCS	1502200				
	Index space exhausted	1580300	x	x		x
	Index resource not available	1570200	x	x		x
	Database/Index inconsistency	1573615	x			x

4.12 USE

4.12.1 Function

This statment specifies the procedures to be followed if the execution of a DML statement results in a data-base-exception condition.

4.12.2 General Format

```
USE FOR DB-EXCEPTION.
```

4.12.3 Syntax Rules

1. The USE statement, when present, must immediately follow a section header in the DECLARATIVES part of the PROCEDURE DIVISION and must be followed by a period. The remainder of the section must consist of one or more paragraphs defining the procedure to be followed.

2. Only one USE FOR DB-EXCEPTION statement may appear in the DECLARATIVES part.

3. Within the definitions of the USE procedure, there must be no reference to non-declarative procedures.

4.12.4 General Rules

1. Appropriate values are placed in the special registers DB-STATUS, DB-REALM-NAME, DB-RECORD-NAME, DB-SET-NAME, DB-KEY-NAME and DB-DETAILED-STATUS before the execution of a data-base-exception section.

2. The data-base-exception section is executed whenever the execution of a statement results in a data-base-exception condition. At the end of the execution of a data-base-exception section, control is returned to the statement following the one that caused the exception, unless a STOP RUN is executed in the DECLARATIVES section.

4.13 AMBIGUOUS REFERENCES TO SUBSCHEMA ENTITIES

4.13.1 General Rules

The COBOL compiler does not allow ambiguous references (explicit or implicit) to subschema entities. An unambiguous reference may be established by either using the alias mechanism of the subschema or by qualification.

4.13.1.1 Name Duplication Among Records

The Data Description Processor (DDPROC) checks that there are no duplicates among subschema data names, except for data items or aggregates that can have the same name within various records.

This lack of uniqueness of data names is solved at compile-time by the following record qualification:

```
... data base data name 1 OF record name 1
```

4.13.1.2 Ambiguous Multiple Subschema References

Within a multiple-subschema COBOL program, a given DML verb may refer to an unnamed current-of-run-unit that the compiler cannot affect to a subschema. In this case, it is necessary to qualify the relevant verbs by means of the *db-internal-name* of the DB-clause.

Furthermore, IDS/II reserved structure names exist, namely DB-CXT, DB-PARAMETERS, DB-REGISTERS (and parts of DB-REGISTERS) that must always be qualified.

4.13.2 DML Forms in a Multiple Schema Context

The different Formats of DML verbs that imply qualification by db-internal-name in a multiple subschema context are listed below:

4.13.2.1 ACCEPT

Format 1

```
ACCEPT identifier-1 FROM db-name CURRENCY .
```

Format 2

```
ACCEPT identifier-2 FROM db-name REALM-NAME .
```

4.13.2.2 ERASE

```
                        [{SELECTIVE}         ]
ERASE WITHIN db-name    [{PERMANENT} MEMBERS] .
                        [{ALL      }         ]
```

4.13.2.3 FIND

Format 1

```
FIND WITHIN db-name DB-KEY is identifier-1 .
```

Format 2

```
                                [         {realm-name-2} ]
FIND CURRENT WITHIN db-name     [ WITHIN  {set-name-3   } ] .
                                [         {key-name-4   } ]
```

4.13.2.4 FINISH

```
FINISH db-name .
```

4.13.2.5 GET

```
GET WITHIN db-name .
```

4.13.2.6 IF

Tenancy

```
        {OWNER }
db-name {MEMBER} .
        {TENANT}
```

Membership

```
set-name is [NOT] EMPTY .
```

4.13.2.7 MODIFY

Format 1

```
MODIFY WITHIN db-name [RETAINING-phrase] .
```

Format 2

```
MODIFY WITHIN db-name ONLY {ALL             } MEMBERSHIP
                           {{set-name--1}...}

                           [RETAINING-phrase] .
```

Format 3

```
MODIFY WITHIN db-name INCLUDING {ALL               } MEMBERSHIP
                                {{set-name--1} ...}

                      [RETAINING-phrase] .
```

RETAINING phrase

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

4.13.2.8 READY

```
        [                                {EXCLUSIVE {RETRIEVAL}} ]
        [                                {          {UPDATE   }} ]
        [                                                       ]
READY [ [db-name] ... USAGE-MODE IS {SHARED     RETRIEVAL } ] .
        [                                                       ]
        [                                {          {RETRIEVAL}} ]
        [                                {MONITORED {UPDATE   }} ]
```

4.13.2.9 USE

```
USE FOR DB-EXCEPTION  [ON {db-name}] .
```

4.14 ACCESS TO THE IDSTRACE FILE

To perform the operations listed below on the IDSTRACE file, the programmer may call the DBCS function H_DML_UCTRACE which allows the user to combine program inFormation and DBCS trace data on the same file.

- **the writing of program inFormation onto the IDSTRACE file**
- the opening of or closing of the IDSTRACE file
- the enabling or disabling of the program and/or the DBCS to write operations onto the IDSTRACE file

The IDSTRACE file is opened by the DBCS at the beginning of a session and closed by the DBCS at the end of the session.

The H_DML_UCTRACE open and close functions allow the user to open the IDSTRACE file before the first READY statement of an IDS session and close it after the last FINISH statement. If these functions are used, the IDSTRACE open and close requests which are submitted by the DBCS are ignored.

Figure 4-2 shows a DML program which is used for opening and closing the IDSTRACE file and for writing program inFormation combined with DBCS trace data.

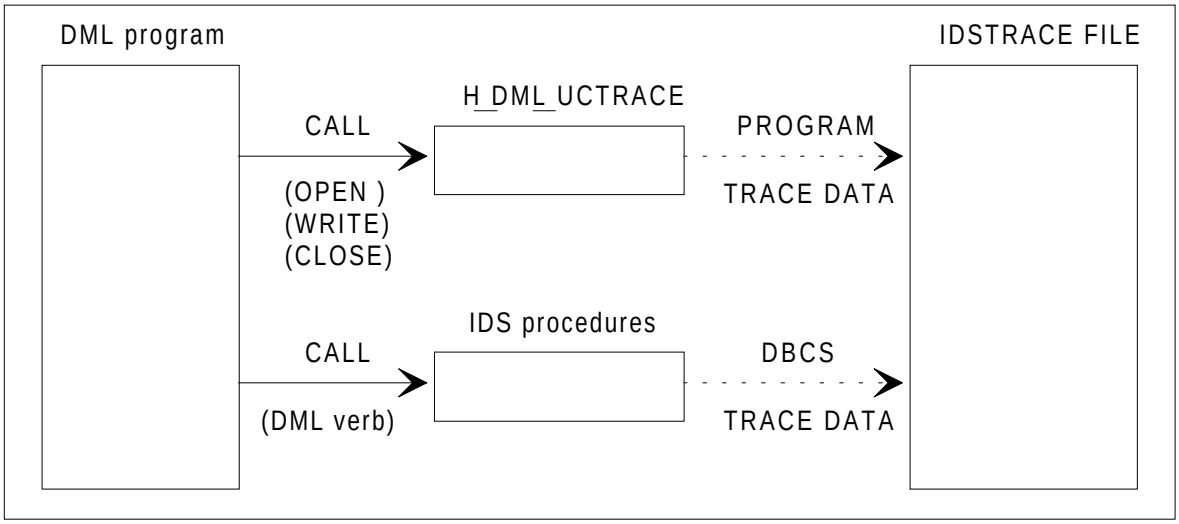


Figure 4-2. User Access to IDSTRACE File (one program)

A run-unit accessing two databases can collect program and DBCS trace inFormation concerning both databases on the same IDSTRACE file. The trace-ifn must be the same for the program and for both databases. The IDSTRACE file is opened by the first open request (submitted by the program or by one of the two IDS sessions) and closed by the last close request.

Figure 4-3 illustrates a COBOL program named P0 which drives two DML programs, P1 and P2, each of which is accessing a different database DB1 and DB2. Program P0 is responsible for opening and closing the IDSTRACE file. P0, P1 and P2 write program inFormation combined with DBCS trace data.

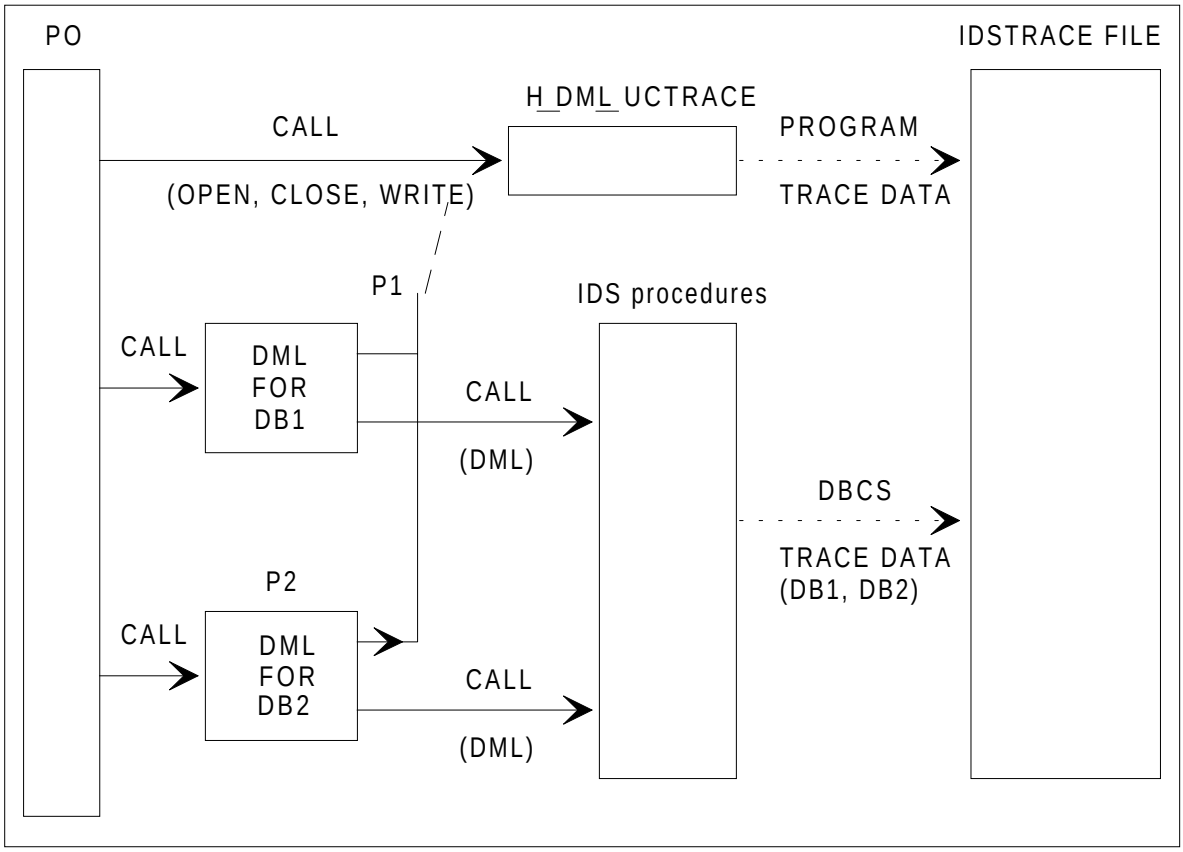


Figure 4-3. User Access to IDSTRACE File (several programs)

The H_DML_UCTRACE disable and enable functions allow the programmer to focus the trace action on a given portion of the program and to apply more specific selection criteria than the TRACE command in the run-unit command language.

4.14.1 Format of Call

```
CALL "H_DML_UCTRACE" USING TRACEPARAM TRACEIFN
```

4.14.2 Parameters

```
01 TRACEPARAM.

02 TRACEFUNCTION COMP-1. (input)
02 TRACERETCODE COMP-1. (output)
02 TRACELENGTH COMP-1. (input)
02 TRACEDATA PIC X(n). (input)
01 TRACEIFN PIC X(8). (input)
```

4.14.3 Parameter Descriptions

TRACEFUNCTION: This parameter is a function code. Values are shown below in decimal.

VALUE	MEANING
1	Disable program write operations onto IDSTRACE file
2	Disable DBCS write operations onto IDSTRACE file
3	Disable program and DBCS write operations onto IDSTRACE file
4	Enable program write operations onto IDSTRACE file
5	Enable DBCS write operations onto IDSTRACE file
6	Enable program and DBCS write operations onto IDSTRACE file
16	Write onto IDSTRACE file
17	Open IDSTRACE file in OUTPUT mode
18	Open IDSTRACE file in EXTENT mode
19	Close IDSTRACE file

TRACERETCODE: return code.

VALUE	MEANING
0	The function has been successfully executed or rendered dummy by a previous disable function (write)
1	The function has not been executed due to incorrect parameters or abnormal conditions

TRACELENGTH: This represents the number of characters (n) of TRACEDATA.

TRACEDATA: This represents program data to be written to the IDSTRACE file.

NOTE: TRACELENGTH and TRACEDATA are only required when TRACEFUNCTION = 16. Otherwise, they are meaningless.

TRACEIFN: This is the internal-file-name of the IDSTRACE file.

4.14.4 Rules

1. The disable, enable, write and close functions are not executed if the IDSTRACE file is not currently opened.

2. The open function is not executed if the IDSTRACE file is not currently closed.

3. When the IDSTRACE file is opened, write operations are implicitly enabled.

4. In a TDS environment, only the write function is available. Other functions are not executed.

4.15 SIMULATION OF THE CALC HASHING ALGORITHM

This function determines the relative page number within a range where a CALC record will randomize, assuming there is no page overflow.

4.15.1 Format of Call

```
CALL "H_DML_UCHASH" USING HASHFUNC HASHRANGEDESC
HASHKEYDESC HASHDATA  .
```

4.15.2 Parameters

```
01 HASHFUNC.
      02 HASHRETCODE COMP-1. (output)
02 HASHPAGENUMBER COMP-2. (output)
01 HASHRANGEDESC. (input)
      02 HASHNUMPAGE COMP-2.
02 HASHCALCINTERVAL COMP-1.
02 HASHINTERVALUNIT COMP-1 VALUE 0.
01 HASHKEYDESC. (input)
      02 HASHNUMITEM COMP-1 VALUE n.
02 HASHITEMDESC OCCURS n.
      03 HASHITEMTYPE COMP-1.
03 HASHITEMLENGTH COMP-2.
03 HASHITEMSCALE COMP-1.
03 HASHITEMPTR COMP-2.
01 HASHDATA. (input)
      Key items within record
```

4.15.3 Parameter Descriptions

Numeric values are given in decimal.

HASHRETCODE: This is the Return code, as explained in the chart below:

RETURN CODE VALUE	MEANING
0	Simulation done
1	Incorrect parameter, which may be: -HASHNUMPAGE =0 -HASHCALCINTERVAL = 0 or > 255 -HASHINTERVALUNIT >< 0 -HASHNUMPAGE not multiple of HASHCALCINTERVAL -HASHNUMITEM = 0 or > 256 -total key length > 256 -invalid HASHITEMTYPE or HASHITEMLENGTH -HASHITEMPTR = 0
2	Illegal decimal data in a CALC-key item

HASHPAGENUMBER: This parameter represents the relative page number within the range resulting from the randomization (0 <= HASHPAGENUMBER <= HASHNUMPAGE - 1).

This is calculated as follows: the number of pages of the range HASHNUMPAGE is divided by HASHCALCINTERVAL, giving the number of buckets as quotient. The CALC-key items described by HASHKEYDESC are retrieved from HASHDATA, concatenated and hashed by the HASH instruction, resulting in a 32-bit binary value. The latter is extended to 64 bits with zero fill and divided by the number of buckets. The remainder is multiplied by HASHCALCINTERVAL, which yields the result: HASHPAGENUMBER.

HASHNUMPAGE: The number of pages of the range. It must be a multiple of HASHCALCINTERVAL.

HASHCALCINTERVAL: The number of pages of the CALC INTERVAL.

HASHINTERVALUNIT: This parameter must be zero.

HASHNUMITEM: The number of CALC-key items.

HASHITEMDESC: This parameter represents the repeating group occuring HASHNUMITEM times and describing each key item.

HASHITEMTYPE,
HASHITEMLENGTH: The type and length of the key item as described in the chart below:

Data type	HASHITEMTYPE	HASHITEMLENGTH
UNSIGNED UNPACKED DECIMAL	9	Number of decimal digits (sign excluded) 1
UNSIGNED PACKED DECIMAL	57	
UNSIGNED PACKED-2 DECIMAL	25	
SIGNED UNPACKED DECIMAL	1	
SIGNED PACKED DECIMAL	17	
SIGNED BINARY	3	Number of binary digits (sign excluded) 15 or 31
CHARACTER	0	Number of characters 1

HASHITEMSCALE: The scale of the key item. 0 for a binary or character item +p for a decimal item. The scale is not _ currently used in the hashing algorithm.

HASHITEMPTR: The character position (starting at 1) within HASHDATA where the key item begins.

HASHDATA: The record containing the CALC-key items and possibly other items.

4.15.4 Example

Consider a range of 101 pages, a CALC interval of 1 page and a CALC-key composed of 3 items: PIC XX, PIC 999, and PIC x located at positions 8, 10, 13 in the CALC record.

The last three parameters of H_DML_UCHASH are as follows:

```
01 HASHRANGEDESC.
      02 HASHNUMPAGE COMP-2 VALUE 101.
02 HASHCALCINTERVAL COMP-1 VALUE 1.
02 HASHINTERVALUNIT COMP-1 VALUE 0.
01 HASHKEYDESC.
      02 HASHNUMITEM COMP-1 VALUE 3 .
02 HASHITEMDESC1.
      03 HASHITEMTYPE1 COMP-1 VALUE 0.
03 HASHITEMLENGTH1 COMP-2 VALUE 2.
03 HASHITEMSCALE1 COMP-1 VALUE 0.
03 HASHITEMPTR1 COMP-2 VALUE 8.
      02 HASHITEMDESC2.
      03 HASHITEMTYPE2 COMP-1 VALUE 9.
03 HASHITEMLENGTH2 COMP-2 VALUE 3.
03 HASHITEMSCALE2 COMP-1 VALUE 0.
03 HASHITEMPTR2 COMP-2 VALUE 10.
      02 HASHITEMDESC3.
      03 HASHITEMTYPE3 COMP-1 VALUE 0.
03 HASHITEMLENGTH3 COMP-2 VALUE 1.
03 HASHITEMSCALE3 COMP-1 VALUE 0.
03 HASHITEMPTR3 COMP-2 VALUE 13.
01 HASHDATA.
      02 R01-IDENT PIC X(7).
02 R01-CALC.
      03 R01-CALC1 PIC XX.
03 R01-CALC2 PIC 999.
03 R01-CALC3 PIC x.
      02 R01-NUMBER PIC 9(4).
```

5. COBOL/DML Compiling and Linking

5.1 USING SL AND DD4 OBJECTS

1. COBOL handles two types of object:

   - SL (Source Language) objects which correspond to the COBOL/DML input source and the COBOL output source

   - DD4 (Data Description) subschema objects which correspond to the object subschema used in DML processing

2. The COBOL/DML input source can be located in either:

   - an input-enclosure with a TYPE option of COBOL or DATASSF

   - or a member of an SL library. The TYPE option of the member must be COBOL, COBOLX or DATASSF.

3. The object subschema is a member of a BIN library.

4. Libraries containing the object subschema must be assigned statically by using the keywords/ifn's DDLIB1, DDLIB2, DDLIB3.

   The member containing the subschema whose name is referenced in the SUB-SCHEMA section is searched in the libraries assigned, in the sequence defined by DDLIB1, DDLIB2, DDLIB3.

5. The object subschema used by COBOL must have been validated against its own schema.

6. The compiled program contains the name and validation date of the subschema which is used in processing.

   At run-time, this name and validation date must match the name and validation date of the subschema loaded for execution.

5.2 COBOL EXTENDED JCL STATEMENT

COBOL can be called using the following specific options of the extended JCL statement to compile a COBOL program containing DML statements.

```
COBOL
          DDLIB1=(library-description)
      [DDLIB2=(library-description)]
      [DDLIB3=(library-description)]
                    [{NDDLIST}];
                  [{DDLIST }]
```

COBOL-specific parameters are defined below:

DDLIBi — These keywords specify the libraries containing the object subschema and their search path.

DDLIST
NDDLIST — This parameter indicates whether or not the user wants a listing of the description of all used records. The default value is NDDLIST, which means that no listing of the description will be produced.

5.3 COBOL COMPILATION

Compilation of the COBOL source is performed by the COBOL compiler (COBOL). The only special consideration for IDS/II is that the keyword LEVEL=L64 must be specified in the COBOL JCL statement.

Except for the specific options described above, the COBOL extended JCL statement is unchanged. For further details, consult the *COBOL 85 User's Guide*.

5.4 LINKING

After successful compilation of the COBOL/DML program, the user must execute LINKER to produce a load-module. The normal linking procedure is explained in the *LINKER* manual, but linkage of programs containing DML may require that a command statement stream be provided to the linkage editor.

The LINKER statement contains the following parameter groups:

```
{           {*input-enclosure-name              } }
{ COMFILE = {                                   } }
{           {(sequential-input-file-description)} }
{                                                 }
{ COMMAND = '{command} ... '                      }
```

through which LINKER commands are supplied.

The LINKER commands that may be necessary are:

```
SEMPOOL=(NUMSN = +p),
STACK1=(INITSIZE = 4K, MAXSIZE = 64K)
```

The first command mentioned above is necessary if the number of semaphores required by UFAS is greater than that reserved by default by LINKER. The user is informed of this situation by a message on the JOR at execution of a READY STATEMENT (TASKM, ENTRYOV). In the above-mentioned command, the value "p" should be set to 7 + 4 * number-of-data-base-files. The "+" sign is mandatory and does not have the same meaning as a space. In order to avoid computation, the SEMPOOL command may be written as follows:

```
SEMPOOL=(NUMSN = 225),
```

which takes the maximum possible number of semaphores.

The second command is related to the number of stack overflows which occur at execution time. This number must be checked in the JOR, especially when the database is opened in update. If its value is higher than 200, the STACK1 command must be included to increase the initial size of the stack of ring 1 (default value 2K). If 3K is not sufficient, higher values can be chosen.

5.5 EXAMPLES

Let us consider an object subschema INVENTORY-SUBSET and an object schema INVENTORY which exist in the corresponding member of library MIDS.BINLIB.

The user wishes to compile a COBOL/DML program named CONTROL17 against this subschema. The program has the following outline:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CONTROL17.
DATA DIVISION.
SUB-SCHEMA SECTION.
DB db-internal-name USING INVENTORY-SUBSET WITHIN      |
INVENTORY.
                .  .  .
PROCEDURE DIVISION.
                .  .  .
                READY.
                .  .  .
                FINISH.
                .  .  .
          END COBOL.
```

The input source is in member CONTROL17 of library MIDS.SLLIB. The compile unit produced will be in library MIDS.CULIB. All the libraries are assumed to be cataloged.

```
$JOB DMLCOB USER=MURIEL PROJECT=LONGEOT;      |
LIB SL INLIB1=MIDS.SLLIB;

COBOL SOURCE=CONTROL17
      DDLIB1=MIDS.BINLIB
      CULIB=MIDS.CULIB
     LEVEL=L64 MAP XREF DDLIST;

$ENDJOB;
```

The user now prepares a load-module, without special options, in the library MIDS.LMLIB.

```
$JOB LINK USER=MURIEL PROJECT=LONGEOT;
LIB CU INLIB1=MIDS.CULIB;

  LINKER CONTROL17
      OUTLIB=MIDS.LMLIB;

  $ENDJOB;
```

After a number of runs of the load-module, the JOR indicates that the number of stack overflows is excessively high. The user can correct the situation by relinking with the STACK1 command, as shown below:

```
$JOB RELINK USER=MURIEL PROJECT=LONGEOT;      |
LIB CU INLIB1=MIDS.CULIB;

LINKER CONTROL17
    OUTLIB=MIDS.LMLIB
    COMMAND='STACK1=(INITSIZE=4K,
                  MAXSIZE=64K) ';

$ENDJOB;
```

6. User Program Execution

6.1 RUN-TIME ENVIRONMENT

6.1.1 Prerequisites

The following operations are prerequisites to the execution of DML programs.

- the Schema DDL and DMCL must already have been translated into an object schema.
- the subschema must already has been translated into an object subschema and validated against its schema.
- the DML programs must already have been compiled against this subschema and linked.

In order to ensure that the subschema referenced at execution is the same as the subschema referenced at compilation and that the Schema DMCL referenced at preallocation is the same as the DMCL referenced at execution, the consistency checks described below are performed at run-time:

1. Once the run-unit session has been triggered, the subschema name and translation date recorded in each program by COBOL are compared with the subschema name and translation date of the object subschema used at execution. An abort will result if the names, dates or times between these two versions are different. Even though the correct subschema has been loaded, differences between dates/times indicate one of the following inconsistencies:

   - that the subschema has been retranslated and the program not recompiled,
   - that the subschema has not been revalidated after a schema change,
   - that a previous subschema has been saved and loaded but that the program has been compiled with a more recent version of this subschema.

   Note that the date and time compared are those of the subschema translation (not the subschema validation).

2. Upon execution of each READY statement, the schema name and DMCL reference date which are recorded in the label of each area by PREALLOC are compared with the schema-name and DMCL reference date of the object schema used at execution.

6.1.2 Execution Modes

DML programs may be executed in BATCH, IOF or TDS environment:

- The BATCH environment includes jobs that either do not activate the telecommunication lines or that indirectly activate them through the input and output queues of MCS.
- The IOF environment is the group of jobs connected to a terminal under the supervision of the Interactive Operation Facility.
- The TDS environment is the group of steps running under the supervision of the Transaction Driven System.

NOTE: The database utilities DBUTILITY and DBREORG are considered as user steps launched by the same type of basic JCL and able to run in either a BATCH or IOF environment.

6.2 OBJECTS REFERENCED AT RUN-TIME

As far as IDS/II is concerned, a user step references six types of objects:

- Database object schemas
- Database object subschemas
- Database storage areas
- Database index files
- An optional IDSOPT file containing the run-time commands
- Optional IDSTRACE files which collects program and DBCS trace information

6.2.1 Database Object Schemas and Subschemas

Object schemas and subschemas must reside in libraries of type BIN. These libraries must be assigned statically using the reserved ifn's DDLIB1, DDLIB2, DDLIB3. A schema and all the subschemas which reference that schema must be stored in the same library.

If a subschema name recorded in a program which is part of a step is not qualified in the run-time commands, the object subschema is searched for within the libraries assigned, in the sequence defined by DDLIB1, DDLIB2, DDLIB3.

If a subschema name is qualified in a run-time SUBSCHEMA command by the efn of a library or by keywords DDLIB1, DDLIB2 or DDLIB3, the object subschema is searched for within the library specified.

If two subschemas with identical names are used in the step, they must be stored in two different libraries and at least one must be qualified in the run-time commands.

Object schemas required at execution must be in the "DDL-DMCL" state. Object subschemas required at execution must have been validated against their own schemas.

6.2.2 Database Storage Areas

Storage areas must be assigned statically using the ifn's defined in the DMCL or redefined in the run-time commands. Only the areas which are to be put in the READY state need be assigned. If several databases are referenced in the same step, all area ifn's must be different.

6.2.3 Database Index Files

Index files must be assigned statically using the ifn's defined in the DMCL or redefined in the run-time commands. These files need not be READYed or opened because they are automatically opened at the beginning of the run-unit session in the less restrictive usage-mode derived from the ACCESS parameter of the ASSIGN statement. If several databases are referenced in the same step, each corresponding index file must have a different ifn.

6.2.4 The Run-time Command File

The run-time command file can be in the following forms:

- an input-enclosure with TYPE option DATA or DATASSF
- a member of an SL library with TYPE option DATASSF
- a sequential file

The run-time command file, when present, must be assigned statically using the reserved ifn IDSOPT.

The absence of the run-time command file is indicated in the JCL by the absence of the ASSIGN IDSOPT statement. If the run-time command file is absent, the run-time default options apply.

If several databases are involved at execution, the set of run-time commands related to a specific database is identified by the first command in the set, which must be the SCHEMA command.

6.2.5 The IDS Trace Files

IDS trace files are standard or permanent SYSOUT files. They are used only when TRACE commands are specified in the run-time command language or when DML programs call the DBCS function H_DML_UCTRACE.

The ifn which is used to reference a trace file can be one of the following:

- the default ifn IDSTRACE, if it is not redefined in the TRACE command
- the ifn specified in the IDSTRACE clause of the TRACE command
- the ifn specified in the second parameter of H_DML_UCTRACE

A static ASSIGN is only needed when the TRACE output is not directed to the standard SYSOUT.

If several databases are involved, the TRACE output can be directed to different SYSOUT files if the ifn's are different. If the ifn's are the same, the TRACE output can be directed to the same SYSOUT file.

In a TDS environment, any output to an IDS trace ifn is automatically redirected to a member of the library assigned to the TDS reserved ifn DBUGFILE.

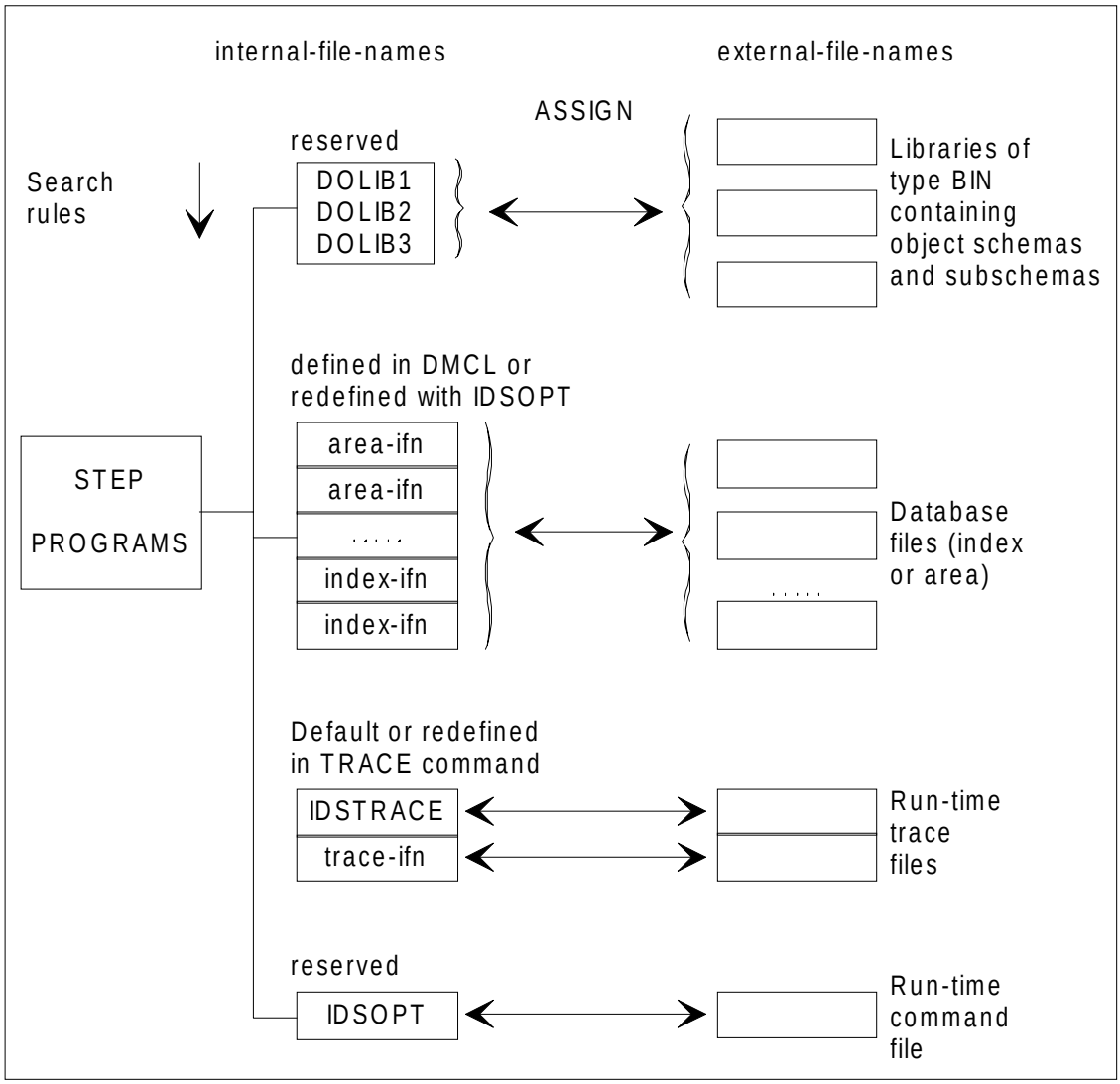


Figure 6-1. Objects Referenced at Run-time

6.3 USER STEP BASIC JCL

The step which starts IDS programs uses the following basic JCL:

```
STEP user-load-module {step-parameters} ... [REPEAT] ;
 [SIZE ...;]
  ASSIGN DDLIB1 library-description;
 [ASSIGN DDLIB2 library-description;]
 [ASSIGN DDLIB3 library-description;]

 { ASSIGN area-ifn area-efn assign-parameters; }
 {    [DEFINE area-ifn define-parameters;]     } ...

 { ASSIGN index-ifn index-efn assign-parameters; }
 {    [DEFINE index-ifn define-parameters;]      } ...

 [ASSIGN IDSOPT {*input-enclosure-name    } ;]
 [   {input-file-description              } ]

 [ASSIGN indstrace-ifn simple-file-description ; ]
 [other statement for non-IDS file ;  ] ...

ENDSTEP;
```

6.3.1 JCL Parameter Description

REPEAT	This parameter is mandatory if the step runs in concurrent access environment (GAC) or if it activates the checkpoint function.
SIZE NBBUF	(total number of buffers) and POOLSIZE (total size of the UFAS pool) must be specified if default values are insufficient.
DDLIBi	These reserved ifn's reference the libraries containing the object schemas and subschemas and also define their search path.
area-ifn	This parameter represents the internal-file-name of a storage area as defined in the Schema DMCL.
An ASSIGN	statement is required for each area involved in processing. This statement specifies the area sharing mode and the processing mode (SHARE, ACCESS parameters) which confirm the usage mode specified in the READY statement.
A DEFINE	statement may be necessary to specify the JOURNAL and READLOCK parameters.

index-ifn	This parameter represents the internal-file-name of an index file as defined in the Schema DMCL. An ASSIGN statement is required for each index file involved in processing. This statement specifies the file sharing and processing mode (SHARE, ACCESS parameters). A DEFINE statement may be necessary to specify the JOURNAL and READLOCK parameters.
IDSOPT	This reserved ifn references the file containing the run-time commands, if any.
idstrace-ifn	This ifn references a SYSOUT file onto which trace information is written. The default ifn is IDSTRACE. It can be redefined in the TRACE command of the run-time command language. Under TDS, the trace ifn cannot be DBUGFILE. The ASSIGN statement is only required if the trace file is not the standard SYSOUT.

6.3.2 Example

Consider, as an example, a schema which has the following DMCL entries:

```
SCHEMA NAME IS MRK-TECH-PUB ...
AREA NAME IS SOFTWARE-MANUALS
    AREA INTERNAL FILE NAME IS SOFTPBS ...
AREA NAME IS HARDWARE-MANUALS
    AREA INTERNAL FILE NAME IS HARDPBS ...
INDEX NAME IS INDEX-MANUALS
    INDEX INTERNAL FILE NAME IS INDEXPBS ...
```

The schema and subschema library, the two areas and the index file are catalogued with the external-file-names DB.BINLIB, DB.SOFT, DB.HARD and DB.INDEX respectively.

Run-time statistics are required and any STORE statements which involve a minimum of ten input-output transfers between the buffers and the database must be traced by the DBCS onto the standard SYSOUT.

If the areas are used in MONITORED mode and protected by the Before and After Journals, the step enclosure may be coded as follows:

```
STEP ... REPEAT;
ASSIGN DDLIB1 DB.BINLIB;
ASSIGN SOFTPBS    DB.SOFT      SHARE=MONITOR  ACCESS=WRITE;|
DEFINE SOFTPBS    JOURNAL=BOTH;
ASSIGN HARDPBS    DB.HARD      SHARE=MONITOR  ACCESS=WRITE;|
DEFINE HARDPBS    JOURNAL=BOTH;
ASSIGN INDEXPBS  DB.INDEX    SHARE=MONITOR  ACCESS=WRITE;|
DEFINE INDEXPBS  JOURNAL=BOTH;
ASSIGN IDSOPT      *RUNCOM;
ENDSTEP;

$INPUT RUNCOM;
SCHEMA NAME IS MRK-TECH-PUB.
STATISTICS WITH TIMING.
TRACE STORE
    WHEN SUBSCHEMA MRK-TECH-PUB-SUBSET
    THRESHOLD IS 10 I-0
    PRINT CURRENT FIELDS STATISTICS TIMING.
$ENDINPUT;
```

6.4 CONCURRENT ACCESS AND FILE PROTECTION

6.4.1 Database Sharing

Three main concepts are involved in the sharing of a database:

1. the resource to be shared: this is the database file at a global level or the database file page at a lower level.

2. the user of a resource: this is the BATCH/IOF step or the TDS transaction.

3. the time period during which a user is granted access to a resource: this is the time delimited by the ASSIGN-DEASSIGN functions at the database file level, or a part of a commitment-unit at the page level.

A database is composed of several storage areas and index database files. If different users access different database files at the same time, they are not considered as sharing the database in the context of the following discussion.

6.4.1.1 Sharing at file level

Three sharing modes are available at file level:

- EXCLUSIVE mode: in this mode, the database file is dedicated to one user who can perform RETRIEVAL or UPDATE operations.

- SHARED mode: in this mode, the database file is shared by several users who perform only RETRIEVAL operations.

- MONITORED mode: in this mode, the database file is shared by several users who perform RETRIEVAL or UPDATE operations. For these users, access control takes place at the page level.

When a user is granted access to a database file in MONITORED mode, he remains a tenant of this resource until DEASSIGN.

6.4.1.2 Sharing at page level

Three sharing modes are available at the page level (when the sharing mode is MONITORED at the file level):

- EXCLUSIVE mode: in this mode, the page is dedicated to **one user** who can perform RETRIEVAL or UPDATE operations.

- SHARED mode: in this mode, the page is shared by several users who perform only RETRIEVAL operations.

- STATISTICAL mode: in this mode, access control is disabled. A copy of the disk page is made available to any user as soon as his request is issued. Only RETRIEVAL operations are allowed in this mode.

When a user is granted access to a page, he remains a tenant of this resource until the end of the commitment-unit.

6.4.2 Usage Modes

In the preceding paragraph, sharing has been considered from the database point of view. In this paragraph, it is considered from the user point of view.

When a user requests access to an area, he indicates the usage intended for this area. Possible usages include the sharing mode at the file level (EXCLUSIVE, SHARED, MONITORED) and the access mode (RETRIEVAL, UPDATE). When sharing at the file level is MONITORED, the usage mode includes sharing and access at page level.

Figure 6-2 illustrates the various possible usages and the different places in the system where they can be specified.

GENERIC USAGE-MODE		**DML TERMINOLOGY READY (FILE)**		**BATCH/IOF TERMINOLOGY JCL ASSIGN (FILE) JCL DEFINE (PAGE)**		**TDS TERMINOLOGY JCL ASSIGN (FILE) TDS GENERATION (PAGE)**	
FILE LEVEL	PAGE LEVEL	FILE LEVEL	PAGE LEVEL	FILE LEVEL	PAGE LEVEL	FILE LEVEL	PAGE LEVEL
EXCLUSIVE RETRIEVAL	////////	EXCLUSIVE RETRIEVAL	////////	SHARE = NORMAL ACCESS = SPREAD	/////////		////////
EXCLUSIVE UPDATE		EXCLUSIVE UPDATE		SHARE = NORMAL ACCES = WRITE			
SHARED RETRIEVAL		SHARED RETRIEVAL		SHARE = NORMAL ACCESS = READ			
MONITORED RETRIEVAL	EXCLUSIVE RETRIEVAL	MONITORED RETRIEVAL	cannot be specified	SHARE = MONITOR ACCESS = READ	READLOCK = EXCL	SHARE = MONITOR ACCESS = {READ} {SPREAD} {ALLREAD}	default EXCLUSIVE UPDATE
	SHARED RETRIEVAL		cannot be specified		READLOCK = NORMAL		TRANSCT. SHARED READ
	STATISTIC RETRIEVAL		cannot be specified		READLOCK = STAT		TRANSACTSUPPRESS CONCURRENT
MONITORED UPDATE	EXCLUSIVE RETRIEVAL	MONITORED UPDATE	cannot be specified	SHARE = MONITOR ACCESS = WRITE	READLOCK = EXCL	SHARE = MONITOR ACCESS = {WRITE} {SPWRITE}	default EXCLUSIVE RETRIEVAL
	EXCLUSIVE UPDATE		always EXCLUSIVE UPDATE		always EXCLUSIVE UPDATE		always EXCLUSIVE UPDATE
	SHARED RETRIEVAL		cannot be specified		READLOCK = NORMAL		TRANSACT. SHARED READ

Figure 6-2. The Designation of Area Usage Mode in Various Contexts

(See explanation of the data in Figure 6-2 below)

The meaning of data in Figure 6-2 is given below:

1. **The first column** lists the generic usage modes as defined above.

2. **The second column** shows the specification of the usage mode in the DML program. It can be seen that only the sharing and access modes at the file level can be specified in the READY statement.

   The logic of the program must be adapted to the usage mode selected (for instance, splitting into commitment-units in MONITORED mode).

3. **The third column** shows the specification of the usage mode in the JCL of a BATCH/IOF step.

   The SHARE and ACCESS parameters of the ASSIGN statement define the sharing and access modes at the file level. The READLOCK parameter of the DEFINE statement defines the sharing mode of pages accessed for retrieval. Pages accessed for update are always reserved in EXCLUSIVE mode.

   File Management takes into account the ASSIGN statement (but not the READY statement) when evaluating the condition for granting access to a file. Therefore, the ASSIGN statement must indicate either the same usage mode as that of the READY statement or a more restrictive usage mode, for example: EXCLUSIVE instead of SHARED or MONITORED, SHARED instead of MONITORED, UPDATE instead of RETRIEVAL. The overriding rules are described in the next paragraph.

   Specifying the usage mode at the JCL level has the following advantages:

   - Job Management is able to avoid deadlocks at the file level.
   - Information concerning the usage mode at the page level can be supplied.
   - The EXCLUSIVE mode at the file level may be defined in the JCL for a program whose logic corresponds to the MONITORED mode. This avoids the overhead of the access control at the page level each time the program runs in this EXCLUSIVE mode.

   Specifying the usage mode at the JCL level has the following disadvantages:

   - The user is led to supply the same information twice, when the usage mode is the same in the program as in the JCL.
   - The ASSIGN/DEASSIGN time period may comprise several READY/FINISH sessions with different usage modes. The ASSIGN and DEFINE statements must then specify the most restrictive of these usage modes.
   - Job Management is able to avoid deadlocks at the file level.

4. **The fourth column** shows the specification of the usage mode in a TDS environment. In the TDS environment, the following elements must be taken into account:

   - The TDS step as a whole is a user with respect to the other BATCH/IOF/TDS steps.
   - The TDS step corresponds to several users represented by the transactions in progress.

The usage mode at the file level is defined in the JCL of the TDS step and applies to all transactions.

EXCLUSIVE mode is not relevant since transactions are usually concurrent.

SHARED mode is not supported for TDS-controlled files.

In MONITORED mode, it is possible to specify the following criteria a file:
- that it is reserved in EXCLUSIVE mode with respect to other steps but is **shared** by the transactions. (This aspect of the SHARED mode is indicated by the ACCESS parameter (SPREAD, SPWRITE)
- that it is shared by other steps as well as by the transactions (READ, WRITE).

The usage mode at the page level is defined in the TDS generation phase and may be different between transactions. The READLOCK parameter of the DEFINE statement must not be specified.

6.4.3 ASSIGN/READY Usage Mode Overriding Rules

The JCL ASSIGN statement can override the program READY statement. The overriding rules are enforced by both File Management and by the DBCS.

Figure 6-3 shows the JCL ASSIGN SHARE and ACCESS parameters that can be specified depending on the READY USAGE-MODE parameter.

When the combination of parameters is not allowed, the intersection of a row and a column contains a hyphen.

When the combination of parameters is allowed, the intersection contains the usage that is retained.

SPREAD, ALLREAD and SPWRITE are not given in Figure 6-3 for the sake of clarity.

If the user wants to define EXCLUSIVE RETRIEVAL in the READY statement, the ASSIGN must specify SHARE=NORMAL and ACCESS={WRITE,SPREAD}.

PROGRAM READY **JCL ASSIGN**		**USAGE MODE**				
		EXCLUSIVE		**SHARED**	**MONITORED**	
		RETRIEVAL	UPDATE	RETRIEVAL	RETRIEVAL	UPDATE
NORMAL	READ	-	-	SHARED RETRIEVAL	SHARED RETRIEVAL	
	WRITE	EXCLUSIVE RETRIEVAL	EXCLUSIVE UPDATE	EXCLUSIVE RETRIEVAL	EXCLUSIVE RETRIEVAL	EXCLUSIVE UPDATE
MONITOR	READ	- -	- -	- -	MONITORED RETRIEVAL	- -
	WRITE	- -	- -	- -	MONITORED RETRIEVAL	MONITORED UPDATE
ONEWRITE	READ	- -	- -	- -	SHARED RETRIEVAL	- -
	WRITE	EXCLUSIVE RETRIEVAL	EXCLIUSIVE UPDATE	SHARED RETRIEVAL	SHARED RETRIEVAL	EXCLUSIVE UPDATE
FREE	READ	-	-	-	-	-
	WRITE	-	-	-	-	-

Figure 6-3. Area Final Usage-Mode

6.4.4 File Protection by Journals

- Two protection mechanisms are provided to restore a database file (area or index), or a part of a database file: the **Before Journal** and the **After Journal**. These mechanisms restore a database file to a known state of stability corresponding to one of the following consistency-points:

  - a beginning of step
  - a check-point
  - a commitment-point
  - an end of step

  A consistency-point is termed **local** if it relates to a given user. Other users may still be working on other parts of the file.

  A consistency-point is termed **global** if it relates to all users; that is, all users have reached a local consistency-point and the database file is inactive.

  The database file protection is based on the assumption that the pages modified by a user between two consistency-points are held in EXCLUSIVE mode.

  By recording the state of the pages before their modification, the Before Journal allows the system to restore these pages to the state they were in at a previous consistency-point. This method is used to recover from program or system failures or to solve deadlock situations.

  By recording the state of the pages after their modification, the After Journal allows the system to restart from a save copy of the file, corresponding to a global consistency-point, and to re-apply all the modifications that occurred until a subsequent consistency-point.

  This method is used to recover from hardware errors causing the physical destruction of the database file. It is therefore recommended that the After Journal not reside on the same volumes as those of the database.

  In a TDS environment, the After Journal may be used in conjunction with the DEFERRED UPDATES mechanism to replace the protection provided by the Before Journal, insofar as there are enough buffers to accomodate all the pages modified during the commitment-unit.

  The usage mode and the operating environment determine whether the Before and After Journals are optional or mandatory. Details are given below:

  a) If the access mode is RETRIEVAL, journals are not used.

  b) If the ASSIGN statement specifies an EXCLUSIVE usage in BATCH/IOF steps, both journals are optional.

  c) If the ASSIGN statement specifies a MONITORED usage with updates in BATCH/IOF steps, the Before Journal is mandatory and the After Journal is optional.

d) If the ASSIGN statement specifies a MONITORED usage with updates in a TDS step, either the Before Journal, or the After Journal with DEFERRED UPDATES must be activated. In this case, if the Before Journal is used, the After Journal is optional.

If several steps update the same database, whether sequentially or concurrently, it is recommended that they activate the file protection journal in a consistent way. This is particularly important for the After Journal when it serves for the protection against media errors.

The DBA can impose a uniform and permanent protection level by specifying JOURNAL options in the catalog.

6.4.5 Index File Usage

Index files (which contain KEY Entries) need not be explicitly READYed by the users. Each environment has an implicit usage mode, as explained below:

- In a BATCH/IOF environment: index files are managed by the DBCS exactly as if a READY usage-mode MONITORED had been issued by the user. If the ACCESS parameter of the ASSIGN statement specifies an access mode WRITE (respectively READ), UPDATE (RETRIEVAL ONLY respectively) will be allowed.

- In a TDS environment: during TDS generation, index files are described in the same way as database areas, and are implicitely READYed according to the same rules as database areas.

6.4.6 Consistency of Protection Levels

As has been mentioned, a given database file must be accessed with the same protection level by several users. It is also important that a user access the different files of the database with consistent protection levels.

The system itself does not perform consistency checks on the protection level of different files. In IDS/II, the database files are part of the same logical structure. If a return to a consistency point is required, the DBCS can run a check to ensure that it is possible to rollback all files which are currently opened.

Figure 6-4 illustrates the constraints which exist when using different files opened in update mode by the same user in the BATCH/IOF environment. In the TDS environment, files must be in MONITORED mode.

The DBA may enforce the consistency of protection levels between files by specifying the same JOURNAL options in the catalog.

<table>
<tr><td colspan="2">Database files already opened
Request for a new database file</td><td colspan="2">EXCLUSIVE UPDATE</td><td>MONITORED UPDATE</td></tr>
<tr><td colspan="2"></td><td>no Before Journal</td><td>Before Journal</td><td>in all cases Before Journal</td></tr>
<tr><td rowspan="2">EXCLUSIVE UPDATE</td><td>no Before Journal</td><td>YES, without ROLLBACK</td><td colspan="2">NO</td></tr>
<tr><td>Before Journal</td><td rowspan="2">NO</td><td colspan="2" rowspan="2">YES
(ROLLBACK possible)</td></tr>
<tr><td>MONITORED UPDATE</td><td>in all cases Before Journal</td></tr>
</table>

Figure 6-4. Consistency of Protection Levels for the same User

6.5 RUN-TIME COMMAND LANGUAGE

6.5.1 Reserved Words

The list of reserved words in the context of run-time command language is given below:

```
ABNORMA              ACCEPT                AFTER
ALL                  ANY                   ARE (IS)
AREA                  (REALM)              AT BEFORE
BIND                 BUFFER (BUFFERS)      CANCEL
CHECK                CLEANPOINT            COMMITMENT
CONNECT              CURRENCIES            CURRENT
DB-REGISTERS         DB-STATUS             DDLIB1
DDLIB2               DDLIB3                DISCONNECT
DUMP                 END                   ERASE
EXCEPT               EXTEND                FIELDS
FILE                 FIND                  FINISH
FOR                  GET                   GETLAB
GETPAGE              GETPTR                HEXADECIMAL (HEXA)
IAC                  IDSTRACE              IF
IGNORE               IMAGE (S)             IN (OF)
INCONSISTENT         INDEX                 INPUT
INTERNAL             IS (ARE)              I-0
KEY                  LINE                  LOCK (S)
MODIFY               MODLAB                NAME (S)
NO                   NON-NULL              NOT
NUMBER               OCCURRENCE            OF
ON-LINE              OPEN                  OUTPUT
PAGE                 POINTERS              POOL
PRINT                PROGRAM               PUTPAGE
READY                REALM (AREA)          RECORD
RELPAGE              RESTART               ROLLBACK
RUN-UNIT             SAVE                  SCHEMA
SEGMENT (S)          SESSION               SET
SOURCE               STARTDBS              STATE
STATEMENT (S)        STATISTICS            STORE
STRUCTURE            SUBSCHEMA             SWITCH
SYNCHRO              SYSTEM                TEMPORARY
TERMDBS              THRESHOLD             THROUGH (THRU)
TIMING               TRACE                 TRANSIENT
UNLOAD               UPDATE                WARNING
WHEN                 WITH
```

6.5.2 Types of Command

There are eleven types of command. The group of commands that correspond to a given schema is identified by the first command in that group, which must be a SCHEMA command.

The run-time command language format is as follows.

```
[ SCHEMA command                ]
[ [IGNORE command]              ]
[ [BUFFER POOL command]         ]
[ [SAVE CURRENCIES command]     ]
[ [SYNCHRO command]             ]
[ [NO WARNING command]          ]
[ [STATISTICS command]          ]
[ [TRACE command]               ]
[ [CHECK PAGE command]          ]
[ [INTERNAL FILE NAME command]  ]
[ [NO ON-LINE command]           ]
```

Run-time commands are described in detail on the following pages of this section.

6.5.3 SCHEMA Command

6.5.3.1 Function

This command identifies the schema to which subsequent commands apply and provides information for the loading of this schema and of associated subschemas.

6.5.3.2 Format

```
                                [      {library-efn} ]
                                [ {IN} {DDLIB1     } ]
SCHEMA NAME IS schema-name      [ {OF} {DDLIB2     } ]
                                [      {DDLIB3     } ]

        [FOR PROGRAM program-name]   .
```

6.5.3.3 Rules

1. Each time a program implicitly triggers an IDS session, the DBCS searches the run-time command file for a SCHEMA command whose schema-name corresponds to that recorded in the program.

2. If the PROGRAM clause is not specified, the search for the SCHEMA command stops as soon as it identifies the right schema-name.

3. If the PROGRAM clause is specified, the DBCS checks that *program-name* corresponds to the name of the program starting the IDS session. The search continues if the names do not match.

   This check is required to differentiate 2 schemas with identical names which could be used by 2 different programs in the step.

4. If the search for the SCHEMA command fails, all default options are applied.

5. If the library qualification is absent, the schema is searched for loading in the libraries assigned, in the sequence defined by DDLIB1, DDLIB2, DDLIB3.

6. If the library qualification is present, the schema is loaded from the library specified.

6.5.4 IGNORE Command

6.5.4.1 Function

This command ignores the transient or inconsistent state of some database files (realms or index) when they are opened.

6.5.4.2 Format

```
        [TRANSIENT ]
IGNORE [INCONSISTENT] STATE OF {realm-name}...
                               {index-name}...  .
```

6.5.4.3 Rules

1. If TRANSIENT is specified, the realms or index of the list are made available to the program even if they are in the TRANSIENT state at opening time.

   The TRANSIENT condition occurs when a previous step which is updating files in EXCLUSIVE mode without Before Journal protection has terminated normally (or abnormally) without finishing the realms or closing the index.

   The TRANSIENT condition indicates that a part of the database is presumably inconsistent. The check of this condition at opening time is a safeguard against the further use of the files.

   The usual way to recover from the TRANSIENT condition is to restore the files from save copies. Two other methods are explained below.

   a) For a **realm** in the TRANSIENT condition: run DBUTILITY to ascertain that the database is consistent, patch local inconsistencies if any, and use the RESET TRANSIENT clause of the PATCH LABEL command.
   b) For an **index** file in the TRANSIENT condition, reconstruct the file by using the BUILD_KEY command of DBUTILITY and use the RESET TRANSIENT clause of the PATCH LABEL command.
      However, if the user is certain that the faulty step left the database files in a **consistent** state (as regards both IDS pointers and user data for a realm for example), he may use the IGNORE TRANSIENT command. The fact that the TRANSIENT state is ignored is recorded in the IDS label.

2. If INCONSISTENT is specified, the realms or index of the list are made available to the program even if they are in the INCONSISTENT state at opening time.

   The INCONSISTENT condition occurs when a previous IDS session has detected and recorded in the IDS label a database inconsistency. The inconsistency is recorded even if the files are used in retrieval mode, except in the following situation:

   when SHARE=MONITOR and READLOCK=STAT, in which case the inconsistency may be only apparent.

   The check of the INCONSISTENT condition is a safeguard against the continued use of the files concerned.

   The usual way to recover from the inconsistent condition is to restore the files from consistent save copies. Two other methods are explained below.

   a) For a **realm** in the INCONSISTENT condition, patch the inconsistent part of the database by using the PATCH command of DBUTILITY after analysis of the DBCS error message which reports the inconsistency; then, validate the patch. If validation is successful, use the RESET INCONSISTENT clause of the PATCH LABEL command.
   b) For an **index** file, reconstruct the file by using the BUILD_KEY command of DBUTILITY and use the RESET INCONSISTENT clause of the PATCH LABEL command.

      However, if the user is certain that the program to be run will not be working on the inconsistent part of the database, he may use the IGNORE INCONSISTENT command. The fact that the INCONSISTENT state is ignored is recorded in the IDS label.

3. If the IGNORE command is omitted, then the program is aborted if it attempts to ready realms or to implicitly open index files which are either in the TRANSIENT or INCONSISTENT state.

4. If the IGNORE command is present, the TRANSIENT and/or INCONSISTENT state of the files concerned will be ignored but not reset.

6.5.5 BUFFER POOL Command

6.5.5.1 Function

This command specifies whether or not certain database files share a buffer pool.

6.5.5.2 Format

```
BUFFER POOL NAME IS

{pool-name                     |AREA {realm-name}...  |
   [WITH integer BUFFERS] FOR  |                      |} ... .
                               |INDEX {index-name}... |
```

6.5.5.3 Syntax Rules

1. A buffer-pool-name must be at most four characters long.

2. If the blank name is used, it must be in the form ' ' (blanks enclosed by apostrophes).

3. Non-blank names may contain letters, digits, the minus sign and the underscore. They must be delimited by apostrophes if they do not conform to the rules for the formation of DDL names.

6.5.5.4 General Rules

1. If the POOL command is absent, then the buffer pool strategy is determined by the relevant clauses specified in the DMCL.

2. If the POOL command is specified, the correspondence between buffer pools and database files is determined by the BUFFER POOL command, which overrides DMCL-supplied information.

3. The minimum value for the number of buffers is 3.

4. If the number of buffers is not specified, the default chosen is 4.

5. If a blank name is specified, the number of buffers equals the number of buffers reserved for each database file object of this command. (Each specified database file has its own set of buffers.)

6. If the same buffer-pool-name is given for more than one database file, a common buffer pool will exist for these files and the number of buffers indicates the total number of buffers that will be shared by these files.

   In a case where different database files have different page sizes, the largest value will be chosen as buffer size.

7. The purpose of a pool of buffers is to reduce the total buffer space requirements when several files are opened. For example, when a DML program accesses one area:

   - if no buffer pool exists, the DML program cannot use more buffer space than that reserved for this area. The space reserved for the other database files is not available.
   - if a buffer pool does exist, the total pool space may be used for this area. When the DML program begins working on a different area, the pool space is assigned to the buffers of the new area.

6.5.6 SAVE CURRENCIES Command

6.5.6.1 Function

This command requests that currency indicators remain **not nulled** across commitment-points in a BATCH/IOF environment.

6.5.6.2 Format

```
SAVE CURRENCIES AT COMMITMENT.
```

6.5.6.3 Rules

1. If this command is omitted, currency indicators are nulled at each commitment-point and the database pages to which they correspond are unlocked by the run-unit.

2. If this command is specified, the currency indicators keep their values across commitments, but the database pages to which they correspond are unlocked by the run-unit at the commitment-point.

   This option should be used only if conventions exist between applications which ensure that the records referenced by the currency indicators of a run-unit are not **erased** or replaced by concurrent run-units. If these conventions do not exist, the DBCS may abort the run-unit if it encounters an inconsistency between the database and the currency indicators.

3. In TDS environment, this command is ignored, since currency indicators are always nulled and the pages unlocked at the commitment-point.

   If the data-base-key of a record is to be kept across commitments, it must be saved by an ACCEPT statement in the TRANSACTION STORAGE before the end of a commitment-unit, and restored by a FIND DB-KEY statement in the next commitment-unit.

4. This command does not apply to check-points. In non-concurrent access environment, currencies are left unchanged across check-points.

6.5.7 SYNCHRO Command

6.5.7.1 Function

This command disables the asynchronous page reading which occurs during the sequential search of an area.

6.5.7.2 Format

```
SYNCHRO.
```

6.5.7.3 Rules

1. If this command is omitted, each "FIND ... WITHIN realm-name" statement triggers the look-ahead mechanism. This mechanism initiates the transfer of the page following (or preceding) the page which is currently needed. This transfer can occur asynchronously while the DBCS or the program is processing the current page, thus reducing the total elapsed time.

2. If this command is specified, the look-ahead mechanism is disabled.

3. This command may be useful when the extra buffer needed for the asynchronous read has to be freed before the page it contains is actually used.

   For example, consider a program using a pool of 4 buffers which sequentially searches an area. For each record found, the program searches a set occurrence spreading over 3 pages in another area. The total number of pages involved in the processing of the occurrence is five: four for the owner and members and 1 for the page read in advance. Since there are only four buffers, the page read in advance for the owner will be released during the processing of the members and will have to be read again for the next occurrence, thus degrading performance. In this case, one of the three methods described below is prefereable:

   - Use the SYNCHRO command
   - Increase the number of buffers
   - Suppress the pool in order to insulate the buffers of the two areas.

6.5.8 NO WARNING Command

6.5.8.1 Function

This command prevents the DBCS from sending warning messages to the JOR (Job Occurrence Report).

6.5.8.2 Format

```
NO WARNING  .
```

6.5.8.3 Rules

1. If the NO WARNING command is not specified, warning messages prepared by the DBCS in the DB-DETAILED-STATUS register are sent to the JOR.

2. If the NO WARNING command is specified, warning messages are not sent to the JOR.

6.5.9 STATISTICS Command

6.5.9.1 Function

This command requests that the DBCS collect statistical and timing information during the IDS/II session.

6.5.9.2 Format

```
STATISTICS [WITH TIMING].
```

6.5.9.3 Rules

1. If the STATISTICS command is specified **without the TIMING option**, the following information is reported in the JOR (Job Occurrence Report) at the end of an IDS/II session:

- the number of buffers used for the each pool
- the number of program calls to each category of DBCS function (ACCEPT, CONNECT, etc.)
- the average number of SYSTEM functions called by each category of DBCS function. SYSTEM functions are limited to UFAS (Unified File Access System) and GAC (General Access Control)
- the average number of page transfers performed by each category of DBCS function (not including I/O for journals)
- the number of calls to each category of DBCS sub-function. Sub-functions are DIRECT PLACEMENT (for the storage of a DIRECT or VIA record), CALC PLACEMENT, SET SELECTION, and SET INSERTION
- the average number of SYSTEM functions called by each category of DBCS sub-function
- the average number of page transfers performed by each category of DBCS sub-function
- the number of calls to each category of SYSTEM functions
- the average number of page transfers performed by each category of SYSTEM function
- the total number of calls to DBCS functions
- the total number of calls to SYSTEM functions
- the total number of page transfers

2. If the STATISTICS command is specified **with the TIMING option**, the following information is also reported:

   - the average elapsed time for each category of DBCS function
   - the total DBCS elapsed time
   - the total DBCS processor time

3. If the STATISTICS command is omitted, no statistical information is collected during the IDS session.

NOTES:

1. Non-integral values are rounded to the closest printable value. For example, an average number of I/O's reported as "0.02" means that the actual value "v" is such that 0.015 < v <= 0.025.

2. The total number of I/O's reported by the DBCS may differ slightly from the number of I/O CONNECTs displayed in the JOR. This is because OPEN and CLOSE I/O's are not available to the DBCS.

3. SYSTEM statistics are intended for the field engineers.

6.5.10 TRACE Command

6.5.10.1 Function

This command activates the tracing of DML statements.

The trace function includes:

- an **external** trace, which helps the user to follow the execution of a DML program. The user can select the types of DML function to be traced, restrict tracing to certain conditions and choose the amount of information to be printed.

- an **internal** trace, which helps the Service Center to check the IDS-UFAS-GAC interface, to check the internal behaviour of the DBCS or to investigate performance problems. The user need not be concerned with the internal trace.

6.5.10.2 General Format

```
TRACE clause
[WHEN SUBSCHEMA clause]
[WHEN RECORD clause]
[WHEN AREA clause]
[WHEN DB-STATUS clause]
[WHEN THRESHOLD clause]
[WHEN PROGRAM clause]
[PRINT clause]
[IDSTRACE clause]
[OPEN clause]
[IAC clause]
```

6.5.10.3 Rules

1. Each Trace command clause is described separately on the following pages.
2. When the internal trace is not activated, the tracing condition is:

```
TRACE function-list-condition
AND SUBSCHEMA-condition
AND RECORD-condition
AND REALM-condition
AND DB-STATUS-condition
AND THRESHOLD-condition
AND (PROGRAM-1-condition [OR PROGRAM-2-condition]
...)
```

3. When the internal trace is activated, the tracing condition becomes:

```
TRACE function-list-condition
AND SUBSCHEMA-condition
AND (PROGRAM-1-condition [OR PROGRAM-2-condition]
...)
```

since the other conditions cannot be evaluated before the end of the DML function.

6.5.11 TRACE Clause

6.5.11.1 Function

This command specifies the DML functions to be traced.

6.5.11.2 Format

```
       { ALL             | ACCEPT     | }
       {                 | CONNECT    | }
       {                 | DISCONNECT | }
       {                 | ERASE      | }
       {                 | FIND       | }
TRACE  { [{ANY EXCEPT}] | GET        | } STATEMENTS .
       { [{NOT       }] | IF         | }
       {                 | MODIFY     | }
       {                 | READY      | }
       {                 | STORE      | }
       {                 | BINDRU     | }
```

```
                {                   | ENDRU       | ]
```

6.5.11.3 Rules

1. If ALL is specified, all DML functions are traced. Besides DML statements, two other functions can be traced:

   - BINDRU (BIND Run-Unit), which is the initialization function triggered by the first READY of a run-unit session.

   - ENDRU (END Run-Unit), which is the termination function triggered by the last FINISH of a run-unit session.

2. The "IF" DML function is the IDS/II name for the Database Condition function.

3. If a list of DML functions is specified where "NOT" (or "ANY" EXCEPT) is not coded, only the functions of the list are traced.

4. If a list of DML functions is specified where "NOT" (or "ANY EXCEPT") is coded, all functions except those of the list are traced.

6.5.12 WHEN SUBSCHEMA Clause

6.5.12.1 Function

This command restricts the trace action to operations through a subschema belonging to a list.

6.5.12.2 Format

```
WHEN SUBSCHEMA NAME IS [NOT] {subschema-name}  ...
```

6.5.12.3 Rules

1. If NOT is absent, the list is made up of the subschemas specified.

2. If NOT is present, the list is made up of all possible subschemas except those specified.

3. If this clause is omitted, the operations are traced whatever subschema is used.

6.5.13 WHEN RECORD Clause

6.5.13.1 Function

This command restricts the action of the trace function to operations which, when execution is completed, place the current record of the run-unit in a list of record-types.

6.5.13.2 Format

```
WHEN RECORD NAME IS [NOT] {record-name} ...
```

6.5.13.3 Rules

1. If NOT is absent, the list is made up of the record-types specified.
2. If NOT is present, the list is made up of all the record-types of the schema except those specified.
3. If this clause is omitted, the operations are traced whatever the type of the current record of the run-unit.

6.5.14 WHEN AREA Clause

6.5.14.1 Function

This command restricts the action of the trace function to operations which, when execution is completed, place the area of the current record of the run-unit in a list of areas.

6.5.14.2 Format

```
      {AREA  }
WHEN {REALM} NAME IS [NOT] {area-name}  ...
```

6.5.14.3 Rules

1. If NOT is absent, the list is made up of the areas specified.

2. If NOT is present, the list is made up of all the areas of the schema except those specified.

3. If this clause is omitted, the operations are traced no matter what area the current record of the run-unit is located in.

6.5.15 WHEN DB-STATUS Clause

6.5.15.1 Function

This command restricts the action of the trace function to operations which, when execution is completed, place certain non-null values in DB-STATUS.

6.5.15.2 Format

```
                        {NON-NULL}
  WHEN DB-STATUS IS     {ABNORMAL}
```

6.5.15.3 Rules

1. If NON-NULL is specified, then the DB-STATUS values are any values but "0000000".

2. If ABNORMAL is specified, then the DB-STATUS values are any values except "0000000","0502100" (end of set, realm or key) and "0502400" (record not found).

3. If this clause is omitted, the operations are traced independently of the DB-STATUS value.

6.5.16 WHEN THRESHOLD Clause

6.5.16.1 Function

This command restricts the operation of the trace function to operations which incur a user-defined minimum number of physical I/O operations (page transfers).

6.5.16.2 Format

```
WHEN THRESHOLD IS integer I-O
```

6.5.16.3 Rules

1. *Integer* specifies the minimum number of I/O operations.

2. If this clause is omitted, the trace function is independent of the number of I/O operations incurred by the DML function being traced.

6.5.17 WHEN PROGRAM Clause

6.5.17.1 Function

This command restricts the operation of the trace function to specified occurrences of DML statements which are coded at specified source line numbers in specified programs.

6.5.17.2 Format

```
WHEN PROGRAM NAME IS program-name

  [SOURCE LINE NUMBER IS integer-1 [THRU integer-2]  ]
  [  [OCCURRENCE IS integer-3 [THRU integer-4]]    ]  ...
```

6.5.17.3 Rules

1. When specified, "integer-1" must be less than "integer-2" and "integer-3" must be less than "integer-4".

2. The specified line numbers ("integer-1", "integer-2") refer to the internal line numbers affected by the COBOL compiler when compiling the specified program.

3. If the SOURCE LINE phrase is not specified, the trace function is activated when the DML function is coded in the program whose name is referenced.

4. If the SOURCE LINE phrase is specified without the OCCURRENCE option, the trace function is activated when the DML function is coded in the program specified, at a source line whose number:

   - either equals integer-1, when the THRU phrase is absent

   - or belongs to the range defined by integer-1 and integer-2, when the THRU phrase is present

5. If the SOURCE LINE phrase is specified with the OCCURRENCE option, the trace function is limited to:

   - the n'th occurrence (n = integer-3) of a DML function within the specified line range, if the THRU phrase is absent

   - the n'th through p'th occurrences (n = integer-3, p = integer-4) of a DML function within the specified line range, if the THRU phrase is present
     The occurrence counter which is related to a line range is set to 0 at the beginning of a run-unit session (BATCH/IOF environment) or at the beginning of a TPR (TDS environment). It is incremented by 1 each time all other trace conditions (TRACE-function-list, SUBSCHEMA, ..., PROGRAM, SOURCE LINE) are satisfied. Specifically, if different DML functions satisfying the conditions are executed sequentially within the same line range, the occurrence counter is incremented for each of them.

6. If several SOURCE LINE phrases are coded, the line ranges must not overlap.

7. If several PROGRAM clauses are specified, they must reference different program-names.

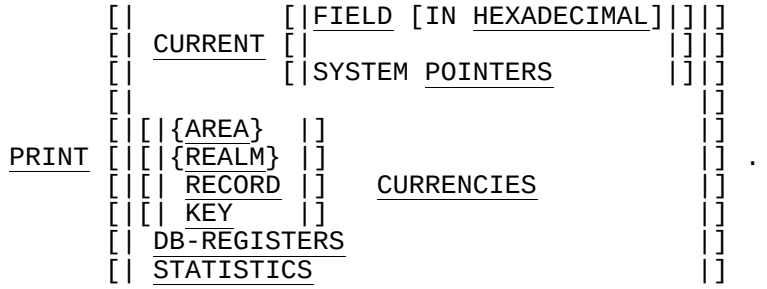
8. If no PROGRAM clause is specified, the operations are traced no matter what program they are coded within.

6.5.18 PRINT Clause

6.5.18.1 Function

This command specifies the amount of information to be printed for each function traced.

6.5.18.2 Format



6.5.18.3 Rules

1. If CURRENT is specified, information on the current record of the run-unit at the completion of the traced function is displayed with the following distinctions:

   - If FIELDS is specified, the data fields of the record are displayed in accordance with their subschema attributes, or in hexadecimal if HEXADECIMAL is coded.

   - If POINTERS is specified, the set pointers of the record are displayed.

   - If neither FIELDS nor POINTERS is specified, both data fields and set pointers are displayed.

2. If CURRENCIES is specified, the contents of the currency indicator at the completion of the traced function are displayed with the following variations:

   - If AREA/REALM is coded, the area currency indicators are displayed.

   - If RECORD is coded, the record-type currency indicators are displayed.

   - If SET is coded, the set-type currency indicators are displayed.

   - If KEY is coded, the key-type currency indicators are displayed.

   - If neither AREA, RECORD, SET nor KEY is coded, all currency indicators are displayed.

3. If DB-REGISTERS is specified, the contents of the database registers at the completion of the traced function are displayed.

4. If STATISTICS is specified, information on the SYSTEM functions activated by the DBCS during the traced function is displayed.

5. If TIMING is specified, the process-time and elapsed-time of the traced function are displayed.

6. If PRINT is specified alone, the maximum amount of information is printed out.

7. If the PRINT is omitted, only the function name and the data-base-key of the current record of the run-unit are displayed.

6.5.19 IDSTRACE Clause

6.5.19.1 Function

This command redefines the internal-file-name of the IDSTRACE file.

6.5.19.2 Format

```
IDSTRACE INTERNAL FILE NAME IS ifn.
```

6.5.19.3 Rules

1. "ifn" must conform to the rules for the formation of internal-file-names as defined in the *JCL Reference Manual*.

   In addition, "ifn" must be different from the following IDS/II values:

   ```
   IDSOPT
   PRTFILE
   COMFILE
   SLLIBa
   DDLIBb
   SAVEc
   ```

   where a,b,c are decimal digits or spaces.

2. If this clause is omitted, the default IDSTRACE is assumed.

6.5.20 OPEN Clause

6.5.20.1 Function

This command specifies the OPEN mode of the IDSTRACE file.

6.5.20.2 Format

```
      {OUTPUT}
OPEN  {      } .
      {EXTEND}
```

6.5.20.3 Rules

1. If OUTPUT is specified, trace information is written starting from the beginning of the file; therefore, previously recorded information is overwritten.

2. If EXTEND is specified, trace information is written starting from the end of the file.

   This option is ineffective if the trace file is SYS.OUT.

3. If this clause is omitted, OPEN OUTPUT is assumed.

6.5.21 IAC Clause

6.5.21.1 Function

This command activates the **internal trace** which displays information on certain system calls performed by the DBCS during the execution of a DML function. IAC stands for Integrated Access Component.

6.5.21.2 Format

```
     [       |   FIELDS [IN HEXADECIMAL]|]
     [       |                          |]
     [       |   SYSTEM POINTERS        |]
     [       |                          |]
IAC  [ WITH  |[| INPUT  |]   PAGE        |] .
     [       |[| OUTPUT|]                |]
     [       |                          |]
     [       |[|INPUT  |]    INDEX       |]
     [       |[|OUTPUT |]                |]
```

6.5.21.3 Rules

1. If IAC is the only parameter coded, the trace function is limited to internal system calls with the corresponding parameters, return-codes, process-time and elapsed-time.

2. If the supplementary phrase FIELDS is coded, the data field values are made available for each storage record involved in an internal call. These are displayed in accordance with their schema attributes, or in hexadecimal if HEXADECIMAL is specified.

3. If the supplementary phrase POINTERS is coded, the set pointers are made available for each storage record involved in an internal call.

4. If the supplementary phrase PAGE is coded, pages accessed and/or modified by the DBCS are printed in hexadecimal.

   - If INPUT is specified, pages requested from Buffer Management are printed before use (internal function GETCI, CHECKCI).

   - If OUTPUT is specified, modified pages are printed before their release to Buffer Management (internal function FREECI).

   - If neither INPUT nor OUTPUT is specified, pages requested from Buffer Management and modified pages released to Buffer Management are printed.

5. If the supplementary phrase INDEX is coded, index entries which are read, stored or deleted by the DBCS are printed in hexadecimal.

   - If INPUT is specified, index entries requested from General Index Manager are printed before use (internal function GETIDX).

   - If OUTPUT is specified, deleted or stored index entries are printed before their release to General Index Manager (internal functions PUTIDX, DELIDX).

   - If neither INPUT nor OUTPUT is specified, entries requested from General Index Manager and deleted or stored entries released to General Index Manager are printed.

6. If the IAC clause is omitted, the internal trace is inhibited.

NOTE: When the IAC clause is specified, the RECORD, AREA, DB-STATUS and THRESHOLD clauses are ignored in the evaluation of the trace condition.

6.5.22 CHECK PAGE Command

6.5.22.1 Function

This comand checks the structure of the pages retrieved or modified by the DBCS.

6.5.22.2 Format

```
      [ [INPUT ] ]
CHECK [ [      ] ] PAGE STRUCTURE .
      [ [OUTPUT] ]
```

6.5.22.3 Rules

1. If this command is specified with the INPUT keyword, the complete structure of pages requested from Buffer Management is validated before use.

2. If this command is specified with the OUTPUT keyword, the complete structure of pages modified from Buffer Mananagement is validated before their release to Buffer Mananagement.

3. If this command is specified with both the INPUT and OUTPUT keywords (or with neither keyword), the complete structure of retrieved and modified pages is validated.

4. If this command is not specified, no validation occurs.

5. The page structure validation is the same as that performed by the VALIDATE PAGE command of DBUTILITY.

6. When inconsistency is detected in the page structure, an error message and the page contents are printed in the JOR and IDSTRACE file, if any, and the step (or transaction) is aborted.

7. This command is intended for the investigation of the cause of database inconsistencies. It should not be used in normal conditions since it consumes processor time.

6.5.23 INTERNAL FILE NAME Command

6.5.23.1 Function

This command redefines the internal-file-names to be used at run-time in the ASSIGN/DEFINE statements referencing the specified database files.

6.5.23.2 Format

```
                      {    { {AREA } realm-name }        }
INTERNAL FILE NAME {FOR { {REALM}               } IS ifn } ... .
                      {    {                     }        }
                      {    {{INDEX  } index-name}        }
```

6.5.23.3 Rules

1. "ifn" must conform to the rules for internal-file-names defined in the *JCL Reference Manual*.

   In addition, "ifn" must be different from the following IDS/II parameters:

   ```
   IDSOPT
   IDSTRACE
   PRTFILE
   COMFILE
   SLLIBa
   DDLIBb
   SAVEc
   ```

   where a,b,c are decimal digits or spaces.

2. If no "ifn" is given for a database file (realm or index), the default which is implicitly or explicitly defined in the DMCL is used.

3. There must be no duplicates among the internal-file-names of a schema.

   If several databases are accessed in the same step and if the internal-file-names are not unique among the different schemas, the user must ensure the uniqueness of each internal-file-name using the INTERNAL FILE NAME command for the relevant schemas.

6.5.24 NO ON-LINE Command

6.5.24.1 Function

This command specifies that a given index file will not be updated at run-time.

6.5.24.2 Format

```
NO ON-LINE UPDATE FOR { index-name }.
```

6.5.24.3 Rules

1. *index-name* must have been declared in the DMCL with the clause ON-LINE NOT MANDATORY.

2. If this command is specified, all modifications concerning a key located in the index file of name *index-name* will be replaced at run-time by a no-operation.

3. When this clause is absent from an index file which contains the clause ON-LINE NOT MANDATORY in the DMCL, "ifn" must conform to the rules for internal-file-names defined in the *JCL Reference Manual*. Key updates will be effective.

6.6 INTERPRETATION OF JOR MESSAGES

6.6.1 Message Format

Except for banners, statistics and syntax error messages, the DBCS formats JOR messages as follows:

```
DDMx ... PGID:y ... SLN:z ... UB:v ... SCH:w ...

                          { ERR }
        ORG:aaa-bbbb ... {     } :u ... Text
                          { WNG }
```

DDM DDM — stands for Data Description and Manipulation. "x..." is a 5-digit decimal number. It is the key identifying the DBCS message text.

PGID — "y..." is a 30-character string containing the program identification as specified in the PROGRAM-ID clause. When this information is not available or is irrelevant, it is replaced by asterisks.

SLN — "z..." is a 7-digit decimal number identifying the source internal line number where the statement involved is coded. When this information is not available or is irrelevant, it is replaced by 0.

SUB — "v..." is a 30-character string containing the subschema name as specified in the SUBSCHEMA section.

SCH — "w..." is a 30-character string containing the schema name characterizing the database on which the current DBCS function has been executed.

ORG — This field provides information on the DBCS module that originated the message.

"aaa" — This is a 3-digit decimal number which identifies a DBCS function. This function may be external (such as STORE) or internal (such as SELECT). The possible function codes are presented later in this section.

"bbbb" — This is a 4-digit decimal number which identifies a point in the procedure code of the DBCS function. This identification depends on the function and type of error and can be used only in conjunction with the listing. It is therefore reserved for the Service Center.

The group "aa bbbb" may be repeated to indicate the stack of DBCS internal functions called when the error was detected and reported. The maximum depth is 10.

ERR/WNG	ERR indicates a fatal error, WNG indicates a warning, "u..." is a 4-digit decimal number and is the code of the error. It is equal to the message key when one message corresponds to one error. It is equal to or greater than the message key when one message covers a whole category of errors. Error codes are given later in this section.
Text	This represents the text in which the error is reported. Complementary information concerning the contents of the text may be found in the *Error Messages and Return Codes Message Directory*.

NOTE: The DBCS messages described above for the JOR can also be sent to the following destinations with the same format:

- to the IDSTRACE file (if one exists),

- to the SYSOUT and TERMINAL reports of the database utilities

- or, for warnings only, placed in the DB-DETAILED-STATUS register of the DML programs.

6.6.2 DBCS Function Codes

DCBS Function Code ranges, which appear in JOR messages, are explained below, and are clearly shown in Figure 6-5. Values are given in decimal.

- Codes xx->yy correspond to external DML statements
- Codes 00->24 correspond to external DML statements.
- Codes 25->29 correspond to prologs generated in COBOL programs.
- Codes 30->34 correspond to the external or internal functions related to the initialization and termination of a session or commitment-unit.
- Codes 35->44 correspond to external functions activated by database utilities.
- Codes 45->49 correspond to internal functions appearing as "sub-functions" in the statistics of an IDS session.
- Codes 50->84 correspond to other internal functions such as those calling UFAS/GAC
- Codes 85->99 correspond to the compatibility layer that converts the 1E/V1 generated code of DML statements.

Function	Code	Function	Code	Function	Code	Function	Code
	00		25	CHECK	50	WAJN	75
ACCEPT	01		26	REPORT	51	OPAJN	76
CONNECT	02	UCINIT	27	ABTERM	52	CLAJN	77
DISCONNECT	03		28	TRACE	53	NOTBJN	78
ERASE	04		29	PAGCHK	54	ROLLBJN	79
FIND	05	BINDRU	30	GETRCALC	55		80
FINISH	06	ENDRU	31	EOPSEM	56	COMPAT	81
	07	ROLLBACK	32	ACCIDX	57		82
GET	08	RESTART	33		58		83
IF	09	CLEANPOINT	34		59		84
	10	GETPTR	35	GETCI	60		85
MODIFY	11	SWITCH	36	FREECI	61		86
	12	PUTPAGE	37	SETCI	62		87
READY	13	GETPAGE	38	LOKC	63		88
	14	RELPAGE	39	CHCKCI	64		89
STORE	15	GETLAB	40	UNLCI	65		90
	16	MODLAB	41	INICMT	66	finish	91
	17	READCMT	42		67	get	92
	18	MODOPT	43		68	if	93
	19	FREELOCK	44		69	modify	94
	20	SELECT	45		70	ready	95
	21	INSERT	46	PGREORG	71	store	96
	22	PLACEDIR	47		72	init	97
	23	PLACECALC	48		73	init1	98
	24		49		74	init2	99

Figure 6-5. DBCS Function Codes in JOR Messages

6.6.3 DBCS Error Code Categories

Figure 6-6, below, shows the possible error codes categories. Values are given in decimal.

The **first category** corresponds to interface problems with the system functions UFAS/GAC/GIM/JOURNAL/TDS. It reports system return codes which the DBCS is unable to interpret or which prevent any further action.

The **second category** corresponds to the analysis of the run-time command language. It includes errors related to the syntax itself and errors due to the IDSOPT environment.

The **third category** corresponds to the errors which occur at the session opening (schema or subschema loading, mismatch between dates, ...) or at the opening of the database file (incorrect ASSIGN, transient state, ...).

The **fourth category** corresponds to the tag and format consistency checks that the DBCS performs continously when accessing its run-time control structures.

The **next six categories** correspond to the consistency checks that the DBCS performs continously when putting together and interpreting the control structures prepared before execution: schema, subschema, program, database. Since it is often difficult to determine which of these elements is faulty, the classification of an error in one of these categories can be arbitrary and may not reflect the actual cause of the error.

The last two categories correspond to warning codes.

CATEGORY	SUB-CATEGORY	ERROR CODE
UFAS/GAC/GIMT /JOURNAL/TDS interface	UFAS/GAC	0 -<=256> 31
	JOURNAL	32 -<=256> 39
	TDS	40 -<=256> 47
	others	48 -<=256> 63
Run-time command language analysis	Syntax analysis	64 -<=256> 351
	Others	352 -<=256> 383
Session or database file initialization	Database session	384 -<=256> 415
	File opening	416 -<=256> 447
	Run-unit session	448 -<=256> 479
	Others	480 -<=256> 511
System consistency checks	Structures tags	512 -<=256> 535
	Structures format	536 -<=256> 559
	TDS environment	560 -<=256>575
	Others	576 -<=256> 767
Database inconsistency		768 -<=256> 895
DML inconsistency		896 -<=256> 1023
Schema inconsistency		1024 -<=256> 1151
Subschema inconsistency		1152 -<=256> 1279
Schema/subschema inconsistency		1280 -<=256> 1535
Schema/Database inconsistency		1536 -<=256> 1791
Schema/DML inconsistency		1792 -<=256> 2047
Warnings		3584 -<=256> 3839
Page structure checks		4096 -<=256> 4351

Figure 6-6. DBCS Error Code Categories in JOR Messages

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 Full IDS/II Reference Manual Vol. 2 Database Products: Full IDS-II

Reference N° :	47 A2 06UDB00	**Date:**	**February 1992**

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 Dept.
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
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Duplicopy@bull.net

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 06UDB00