



Specifications Board

Office des normes

Supply and Services
Canada

Approvisionnement et Services
Canada

CODASYL FORTRAN DATA BASE FACILITY

JOURNAL OF DEVELOPMENT

THIS REPORT HAS BEEN PREPARED BY THE FORTRAN DATA BASE MANIPULATION LANGUAGE COMMITTEE OF THE CONFERENCE ON DATA SYSTEMS LANGUAGES (CODASYL), AN ORGANISATION COMPOSED OF VOLUNTEER REPRESENTATIVES OF COMPUTER MANUFACTURERS AND USERS IN INDUSTRY AND THE FEDERAL GOVERNMENTS OF CANADA AND THE UNITED STATES OF AMERICA.

110-GP-2

January, 1977

Price 4.00

Copies of this document may be ordered from:

Materiel Data Management Centre
Department of Supply and Services
Ottawa, Ont., Canada.
K1A 0S5

Price is \$4.00 per copy, including packing and mailing. Remittances, which should be sent with orders, may be in either Canadian or U.S. Funds and should be made payable to the Receiver General of Canada. When ordering, please quote title precisely as shown on cover.

CODASYL FORTRAN DATA BASE FACILITY

JOURNAL OF DEVELOPMENT

January 1, 1977

Version 1.0
of the
Codasyl Fortran
Data Base Manipulation Language Committee

Address Comments to:
Chester M. Smith, Jr.
214 Computer Building
The Pennsylvania State University
University Park, Pa. 16802

FOREWORD

It is the pleasure of the Conference on Data Systems Languages (CODASYL) to submit this first publication of the FORTRAN DBML Committee. The report reflects the work of some of our most dedicated members and is intended to provide a true FORTRAN-level interface to a Data Base Management System.

The applicability of these specifications reinforces our 1971 decision to divide what was a single Data Base Specification into two general categories:

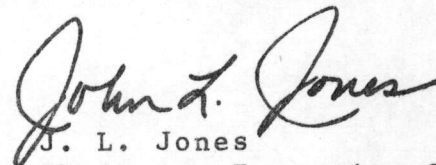
1. elements supporting independent data description; and
2. elements enabling programming languages to interface properly with data described in that independent data description language.

This division has enabled the various CODASYL groups to work with a manageable degree of independence while preserving the capability of gathering the CODASYL Data Description Language (DDL) together with its COBOL Data Manipulation Language (DML) and now its FORTRAN DML to form a coordinated, powerful and machine independent data base management package.

We realize that the task is far from complete and, for this reason, continue to work in these areas as well as those of Data Base Administration, End User Facilities, and Operating System Control Language. Having access to CODASYL-described data bases via both COBOL and FORTRAN widens tremendously the spectrum of systems users possibly served by our specifications. With this in mind, we expect that, as implementation and application of these CODASYL tools widen, interest in modifications, additions and indeed whole new areas of investigation will appear and will continue to provide the incentive to seek solutions.

While these FORTRAN specifications are new, they do represent serious work by industry experts. We therefore invite implementation and application of the concepts and pledge CODASYL's support as a forum to air, negotiate and improve their practicability.

Your comments on and participation in this and all other CODASYL activities are welcomed and sincerely encouraged.



J. L. Jones
Chairman, Executive Committee
Conference on Data System Languages

ABSTRACT

This document summarizes a facility to allow the FORTRAN user to manipulate data bases. It is based on both the CODASYL DATA BASE FACILITY[6] and the draft of the new ANSI FORTRAN standard[1]. This report describes in detail the semiotics of a set of data manipulation language (DML) statements and Sub-schema data definition language (DDL) statements intended to be in the spirit of FORTRAN.

The document is intended to stand alone and to be as complete as possible in specifying a data base facility for FORTRAN.

ACKNOWLEDGMENT AND DISCLAIMER

This Journal of Development is a product of the FORTRAN DBML Committee of CODASYL. As with all CODASYL specifications, this language is considered to be an industry language and not the proprietary property of any individual or organization.

Any organization interested in reproducing this CODASYL report and its specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention "CODASYL" in acknowledgment of the source, but need not quote the acknowledgment in full.

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

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

TABLE OF CONTENTS

1	PREFACE	5
2	DATA BASE CONCEPTS AND NOTATIONAL RULES	7
2.1	DATA BASE CONCEPTS	7
2.2	NOTATIONAL RULES	12
2.3	META-SYMBOLS	16
2.4	RECORD SELECTION EXPRESSIONS	25
2.5	TENANCY CONDITIONS	31
2.6	MEMBERSHIP CONDITIONS	32
2.7	NULLITY CONDITION	33
3	FORTRAN SUB-SCHEMA	34
3.1	SUBSCHEMA STATEMENT	35
3.2	ALIAS STATEMENT	38
3.3	REALM STATEMENT	40
3.4	SET STATEMENT	42
3.5	RECORD STATEMENT	47
3.6	TYPE-STATEMENTS	52
3.7	END STATEMENT	54
4	DATA MANIPULATION LANGUAGE (DML)	55
4.1	ACCEPT STATEMENT	56
4.2	CONNECT STATEMENT	58
4.3	DISCONNECT STATEMENT	60
4.4	ERASE STATEMENT	61
4.5	FETCH STATEMENT	63
4.6	FIND STATEMENT	64
4.7	FINISH STATEMENT	65
4.8	GET STATEMENT	66
4.9	INVOKE STATEMENT	67
4.10	MODIFY STATEMENT	68
4.11	ORDER STATEMENT	70
4.12	PRIVACY STATEMENT	73
4.13	READY STATEMENT	77
4.14	STORE STATEMENT	79
4.15	USE STATEMENT	81
5	EXCEPTION HANDLING	82
5.1	DATA BASE SPECIAL REGISTERS	82
5.2	PROCEDURAL FLOW AFTER DATA BASE EXCEPTION	83
5.3	DATA BASE STATUS INDICATORS	84
6	STATEMENT SYNTAX SUMMARY	88
7	REFERENCES	112
8	GLOSSARY	113
	APPENDIX A - SAMPLE SUB-SCHEMA AND FORTRAN PROGRAM.....	122
	APPENDIX B - FORTRAN EXTENSIONS NOT PRESENTLY IN COBOL DATA BASE FACILITIES.....	126
	INDEX	127

1. PREFACE

The Conference on Data Systems Languages (CODASYL) presents this as the first Journal of Development of its FORTRAN Data Base Manipulation Language Committee. This report is intended to be a complete description of the features necessary to make the facilities of a CODASYL data base system available to the FORTRAN user.

As computer technology evolves, several facts are beginning to dictate the course of this evolution. Included in these facts are that the size of the stored files is growing and the vast majority of the data is not even computerized at this time, but will be as time permits. In addition to this, the cost of magnetically storing data continues to drop as the physical size of the devices available to contain this data dramatically increases. It is also observable that scientific theories often await testing because the data are simply too hard to prepare. It is also disturbing that the needed data are often available, but not in a form that can be adapted to the programming expertise of the interested person. It is a stated objective of a data base system that the programming be kept as simple as possible, and the data in the data base as independent as possible of the programs that use the data.

Additionally to ensure that the data in a data base are put to the best use possible, it is reasonable to have widely-accepted programming languages enhanced with the capability of allowing use of the data. Because so many scientific and engineering programs are written in FORTRAN, and because so much research and development comes from these programs, a group of people interested in enhancing this language with data base capabilities met in 1974 under the auspices of the Conference on Data Systems Languages (CODASYL) and formed a committee to look into such specifications for FORTRAN. This committee has been guided by two general sets of specifications, ANSI FORTRAN and the CODASYL DBLTG Report.

Discussions yielded the specific syntax of commands that take the form of a command verb generally followed by a list of parameters enclosed in parentheses. The parameters are separated by commas, as is common in FORTRAN. A nested series of parameters without parentheses is used in some commands, in the traditional FORTRAN manner. Even though twenty new statements have thus been added to FORTRAN, this method was felt to be a better approach than simply describing a series of procedure calls.

Preserving the recent enhancements to the FORTRAN language, a "keyword" form of command parameter has been used extensively. The decision to use the keyword form, rather than the positional form, is expected to provide more understandable data base command selection. It is believed that this will contribute to the reliability and maintainability of the programs, and facilitate the learning of the command syntax.

The committee adapted the DBLTG semantics to FORTRAN with little modification. Consequently the COBOL and FORTRAN DML statements are purposely quite similar. The Committee decided to define a dynamic naming capability for records, sets, and realms during execution of the FORTRAN program. This will facilitate the programming of on-line interpretative data base systems.

The mechanical difficulties of producing the multiple versions and modified versions of the syntax definition led to a decision to modify it so that the description could be produced and edited by computer. The long vertical braces, long vertical brackets and long vertical bars, that were hand-drawn in previous versions of the syntax definitions, are no longer used in the new version. The short one-line vertical bar to represent the use of "or" has been introduced to provide some of the capabilities supplied by the previous long vertical form. The new definition method has some of the characteristics of Backus Normal Form (BNF) notation and some of the characteristics of the previous braces used in the COBOL documentation. Preliminary common element definitions have been used to simplify the overall commands and to provide improved clarity, as well as to facilitate the physical aspects of the definitions. The overall results are that the syntax definitions can now be prepared and modified by on-line text editors and printing systems.

The committee believes that the order of presentation of this document should not include terms that are already familiar to the reader experienced in data base systems, but that these well known terms and concepts be presented in the glossary. Excellent supplemental information is available in documents listed in Section 7.

Committee members who contributed to the first draft and their respective organizations were:

Chester M. Smith, Jr., Chairman, The Pennsylvania State University
Robert W. Drake, Vice-Chairman, Honeywell Information Systems
John M. Tyler, Secretary, Louisiana State University
Melvin E. Black, Editor, Illinois Power Company
Seth S. Cohen, Digital Equipment Corporation
Thomas J. Harris, Sperry-Univac
John B. Mc Lean, United States Air Force
Kenneth Sperka, Social Security Administration
Wood Rehling, Control Data Corporation

2. DATA BASE CONCEPTS AND NOTATIONAL RULES

This chapter contains essentially those concepts and notational rules that are uniquely pertinent to this document. The remainder of the data base terms, that may be necessary as additional background for less experienced readers, are contained in the Glossary (See Section 8).

2.1 DATA BASE CONCEPTS

2.1.1 Data Base Facility

A data base facility is a generalized software support capability for storing data in, and retrieving data from, a data base. It is activated by or for a user program, and provides control and access in a standardized manner.

The concepts of data control and access have evolved from the earlier notion that each program creates and manipulates its own data, independent of other data that exist in the computer system. The concept of a data base, shared among several programs simultaneously, places additional requirements on the computer system which has access to that data base:

1. Data therein must be accessible in a form and order which is suitable to each individual language.
2. Data therein must be simultaneously accessible to all executing run units.
3. Data integrity must be protected from unfavorable results that could occur during simultaneous access by different run units.
4. Data must be protected from unauthorized access.

The FORTRAN Data Base facility provides access to and control of a data base in accordance with the above requirements.

2.1.2 Data Base Environment

The environment in which a FORTRAN program accesses a data base consists of at least the following:

1. The data base itself, consisting of all the data that are to be accessed.
2. A Data Base Control System (DBCS) that provides the interface to the data base.
3. A Schema which describes the structure of the data base.
4. A Sub-schema which describes that portion of the data base which is to be used by the FORTRAN program, and describes the form in which the FORTRAN program will use those data.

5. A FORTRAN program containing statements and phrases pertaining to the data base.

2.1.2.1 Data Base Control System

The DBCS is that collection of functions which, in conjunction with the operating system, accesses the data base.

2.1.2.2 Schema

The description of the entire data base is known as the Schema. The Schema is written in the language defined in the CODASYL Data Description Language Journal of Development.

2.1.2.3 Sub-schema

The description of that portion of the data base that a given FORTRAN program is to access is known as a Sub-schema. A Sub-schema is written in the FORTRAN Sub-schema language.

2.1.3 Relationship Between Schema and Sub-Schema

Many characteristics of the data in the data base are fixed by the Schema and cannot be changed by the Sub-schema and/or the FORTRAN program accessing the data, i.e. Set Order Selection, Data Base Procedures, etc. Other characteristics can be specified differently in the Sub-schema and the Schema, e.g. the type of a data item. The DBCS performs any necessary transformations of data during retrieval and storage.

FORTRAN Sub-schemas are stored in the Sub-schema Library. A FORTRAN program uses a Sub-schema by specifying its name in the INVOKE statement.

The first statement of a Sub-schema is the SUBSCHEMA statement, and it identifies which Schema the Sub-schema is associated with. Therefore, in a data base environment, there is a unique triplet of a Schema, a Sub-schema, and a FORTRAN program which, when bound together, performs the functions of the user-specified program.

The compilation of the Schema, the Sub-schema, and the FORTRAN program may each occur at different times. Additionally, the binding of these three may also occur at different times.

The following specifies some possible times when binding may take place. The control and availability of these options are implementor-defined.

1. FORTRAN compile time.
2. Sub-schema compile time.
3. Execution time.
4. Any other implementor-defined time.

2.1.4 Records in the Data Base

A record in the data base is conceptually like a record in a file, in that it is a collection of data items. However, the data base record is significantly different from a record in a file in a number of ways. A data base record is not necessarily a physically contiguous collection of data items. In fact, data items that make up the record need not be stored in the data base at all if they are described as virtual derived data items. Regardless of the method of storage in the data base, the DBCS assembles the record as described in the Sub-schema and makes it available to the FORTRAN program, or stores the record generated by the FORTRAN program.

Each Record Description entry in the Sub-schema describes one record type. There is an arbitrary number of occurrences of that record type stored in the data base.

2.1.4.1 Data Base Keys

A data base key is a conceptual entity whose value uniquely identifies an occurrence of a data base record. A data base key is not part of any data base record, but its value may be accessed or used by a FORTRAN program.

When an occurrence of a record type is initially stored in the data base, the DBCS assigns a unique data base key value to the record. The value remains invariable for that record until the record is logically removed from the data base. After that time, the DBCS can reassign that data base key value to a new record occurrence.

2.1.4.2 Realms

A realm is a named logical subdivision of the data base. It is that portion of the data base that contains all records having a specified range of data base key values. Realms are defined in the Schema.

2.1.4.3 Sets, Set Types, and Set Occurrences

The set is a basic structural unit of the data base. A set type is a named collection of record types. One record type is defined as the owner of the set type, and one or more other record types are defined to be members of the set type. In the data base, each occurrence of the owner record constitutes a set occurrence. A member record type, in turn, can be defined to be the owner of other sets of different types. In the data base, a set (an occurrence of an owner record and occurrences of its member records) is identified by the owner record.

Each set type has an order specified for its member records, to permit the DBCS to control the sequence of storage of new member occurrences into the set. A set selection criterion is associated with each set type to aid the DBCS in selecting a specific owner record occurrence and thus a specific set occurrence of that set type.

2.1.4.3.1 Set Ordering

The set ordering criteria defined in the Schema for each set type determine the logical ordering of the occurrences of the member records of that set. The physical order of the member records in the data base is independent of the logical ordering specified by the set ordering criteria specified in the Schema. In addition, the set ordering criteria for each set type can be different, so that a record occurrence that is a member of two different set types could be subject to retrieval in two entirely different orders. The set ordering criteria of a set type can be any of the following:

1. In an order based upon the values of data items appearing in the record's occurrence.
2. In the order of storage of record occurrences, either first in the set or last in the set.
3. Procedurally selected, either immediately prior to or after a record occurrence selected by the FORTRAN program.
4. Any other criteria as specified in the Schema.

2.1.4.3.2 Set Membership

There are two storage classes and two removal classes of set memberships.

2.1.4.3.3 Storage Class of Membership

A record type's storage class of membership in a given set type determines the conditions under which records of that record type become members of sets of that set type. A record's storage class of membership can be either AUTOMATIC or MANUAL. If it is AUTOMATIC, the record automatically becomes a member of that set when the record is initially stored in the data base. If MANUAL, the record can become a member of that set only if it has been previously stored in the data base and a CONNECT statement is applied to it.

2.1.4.3.4 Removal Class of Membership

A record type's removal class of membership in a given set type determines whether or not a record occurrence of that record type can be removed from membership in a set occurrence of that set type, once a member. A record's removal class of membership is independent of its storage class of membership. A record's removal class of membership can be either TRANSIENT or PERMANENT. If it is PERMANENT, the record can only be removed from membership in that set by executing an ERASE statement; if TRANSIENT, the record can be removed from membership in that set by execution of a DISCONNECT or ERASE statement.

2.1.4.4 Record Placement and Selection

When a record occurrence is initially stored in the data base, its logical placement is determined by one of several means. When the record is subsequently retrieved, it is selected and identified for retrieval by one of several means.

2.1.4.4.1 Currency Indicators

Currency indicators are quantities that are maintained by the DBCS to keep track of record retrieval and storage. There is one unique currency indicator for each run unit and, in addition, one for each REALM available to that run unit, one for each set type available to that run unit, and one for each record type available to that run unit. Unless the action is specifically suppressed, the DBCS automatically updates the currency indicators when a record of a given type is retrieved, stored, or modified. The contents of a currency indicator consist of a data base key value that identifies a record.

2.1.4.4.2 Location Modes

Each record type description in the Schema has associated with it a LOCATION MODE that is used by the DBCS in determining how to assign a data base key value to each record occurrence of the specified record type when it is initially stored in the data base. The LOCATION MODE of a record type can specify that the DBCS is to assign a specific data base key value, or a data base key value based on the value of data items in the record, or a data base key value that is closely related to the data base key values of members of a given set.

2.1.4.4.3 Set Selection

Each member type of a set type has associated with it SET SELECTION criteria as defined in the Schema. The SET SELECTION criteria specify the method to be used by the DBCS to select a specific owner record occurrence of a set of that set type, and thus select the member records of that set occurrence.

2.1.4.4.4 Record Selection Expressions

Record Selection Expressions are specified in the FIND/FETCH statements of a FORTRAN program for the purpose of selecting a specific record occurrence in the data base for subsequent use. See Section 2.4.

2.1.4.5 Data Base Procedures

Some of the functions of the DBCS can be controlled by Data Base Administration procedures during the execution of an object program, rather than by the more common declarative control. This procedural control is achieved through the use of data base procedures. A data base procedure is a routine to which control is transferred during the execution of an object program, as defined in the Schema or in the Sub-schema. Note that the data base procedures are not explicitly called by the FORTRAN program, but are invoked automatically by the DBCS.

2.2 NOTATIONAL RULES

In defining the statements described in this report, a special meta-notation is used. The notation resembles Backus Normal Form (BNF) and has been chosen for simplicity of representation. The following paragraphs further describe this notation. The reader should note that the syntax definitions found in this report should be interpreted via the rules for BNF and these three additional high-level rules:

1. The source statements for the FORTRAN Data Base Facility (both Sub-schema and DML) follow the rules for comments, continuation lines, statements and statement labels as defined in FORTRAN [1] Sections 2.3, 3.2, 3.3, and 3.4.
2. In the DML statements the rules for blank characters are as defined in FORTRAN [1] Sections 3.1.6 and 3.3. The Sub-schema DDL statements also follow these rules except for the restrictions that blank characters must not be embedded in words (keywords or identifiers) and adjacent words must be separated by one or more blank characters.
3. A comma that is normally defined in the syntax for a statement may be deleted if it follows immediately after a left parenthesis or another comma, e.g.

ERASE (,PERMANENT) becomes ERASE (PERMANENT)

4. The parentheses enclosing a list may be deleted if the list is singular, e.g.

REALM = (ENG) becomes REALM = ENG

The following paragraphs comprise a brief tutorial on the meta-notation used in this document. Readers familiar with the use of BNF may proceed to Section 3 after briefly reviewing the following paragraphs. If the reader is unfamiliar with BNF, it is suggested that the examples be compared to the syntax rules and that the following paragraphs be reviewed whenever difficulty is encountered.

The definitions found in this document describe syntactically correct statements. The mechanism used for this purpose is to provide a "syntax skeleton" for each Sub-schema and DML statement. A syntax skeleton is formed from a set of syntax component names, delimiters, and keywords, which may be grouped by special notational symbols. The notational symbols denote optional, alternative, mandatory, or repeated sequences which may occur in the FORTRAN source statement. The following conventions apply within a syntax skeleton:

1. Identifiers represent keywords in the source statement; i.e. the occurrence of a keyword is used to establish the context of subsequent syntactic units.

Syntax skeleton: REALM=<db name>

Statement usage: REALM='AB-ACCT'

The syntax skeleton above shows a keyword REALM to be followed by an equal sign and a meta-symbol <db name> phrase. The statement usage of this skeleton shows that the keyword REALM is used to establish the context of 'AB-ACCT' (which has replaced the meta-symbol <db name> phrase). The interpretation of the above statement usage is that 'AB-ACCT' is to be used as the name of a data base realm.

2. Commas, parentheses, and the equality symbol represent corresponding symbols in the source statement (see the previous example).
3. Angle brackets "<" and ">" enclose the name of a meta-symbol. (see the previous example).
4. Brackets "[" and "]" enclose optional clauses. An optional clause is one which may be omitted or occur once, unless repetition is indicated:

Syntax skeleton: STATUS=<integer>[,<integer>]

Statement usage: STATUS=4
 or STATUS=4,9

This syntax shows a keyword STATUS which is to be followed by the equal sign and an integer. The integer may optionally be followed by both a comma and a second integer.

5. A vertical bar "|" separates alternative choices.

Syntax skeleton: INSERT|MODIFY|STORE

Statement usage: INSERT
 or MODIFY
 or STORE

A single selection is to be from the list of alternatives presented.

Syntax skeleton: {RECORD|SET}=<db name>

Statement usage: RECORD='REC-NAME-1'
 or SET='NAME-OF-SET'

The braces in the skeleton are used to enclose the range of alternative choices in order to avoid ambiguity in the syntax skeleton. The braces ("{" and "}") do not actually appear in the statement usage; they are only for use in analyzing the syntax skeleton.

6. Braces "{" and "}" enclose a clause from which exactly one phrase must be chosen (See the previous example).
7. Repetition "..." is used to indicate that the preceding phrase or clause may be repeated one or more times.

Syntax skeleton: STATUS=<integer>[,<integer>]...

Statement usage: STATUS=9
 or STATUS=9,10
 or STATUS=9,10,11,12

This syntax skeleton indicates that an indefinite sequence of integers (separated by commas) may follow the equal sign. Note that the repetition indicator "..." is applied only to the immediately preceding syntactic unit, i.e. the optional clause.

Syntax skeleton: LIST=<integer>...

Statement usage: LIST= 1
 or LIST= 12
 or LIST= 123

This syntax skeleton indicates that the <integer> phrase meta-symbol may be repeated indefinitely, but that it must occur at least once.

Meta-symbols are used to name commonly-used syntax skeleton components or clauses. When the name of a meta-symbol occurs in a syntax skeleton, the rules found in the definition of that meta-symbol are to be applied. The mechanism for defining a meta-symbol is:

<meta-symbol-name> ::= syntax rule

This definition is read, "A meta-symbol-name is defined by a syntax rule". The syntax rules used in the definition of a meta-symbol follow the conventions listed previously.

Syntax skeletons:

```
<retain> ::=
```

```
<retain-1> | <retain-2>
```

```
<retain-1> ::=
```

```
RETAINING = ({ [RECORD] [, REALM]  
               [, SET | {SET=<db names>}} ] )
```

```
<retain-2> ::=
```

```
RETAINING = MULTIPLE
```

Statement usage:

```
RETAINING=(RECORD,REALM)  
RETAINING=(REALM,SET)  
RETAINING=(REALM,SET='ABC')  
RETAINING=(SET=('ABC','DEF'))  
RETAINING=MULTIPLE
```

The first four examples show statement usage of the <retain-1> phrase; the last example is the <retain-2> phrase. Note the difference in the SET=<db name> clause construct, where several names are enclosed in a parenthesized list ('ABC','DEF') and a single name need not be parenthesized: SET='ABC'.

2.3 META-SYMBOLS

The following paragraphs discuss the meta-symbols used in building the rules for the FORTRAN DML statements. Refer to Section 4 for the semiotics of individual statements.

2.3.1 Arguments

<argument> ::=

as defined in FORTRAN [1] Section 15.6.2.3.

2.3.2 Character Constant

<char con> ::=

as defined in FORTRAN [1] Section 4.8.1 with a maximum length not to exceed 30 characters.

2.3.3 Character Expression

<char exp> ::=

as defined in FORTRAN [1] Section 6.2.

2.3.4 Character Variable

<char var> ::=

as defined in FORTRAN [1] Section 8.4.2.

2.3.5 Data Base Key

<db key> ::=

integer variable as defined in FORTRAN [1] 8.4.1 with a non-negative value.

A data base key is a logical "address" which is interpreted by the DBCS to identify a desired record occurrence. The variable holding a data base key is of type integer.

2.3.6 Data Base Name

<db name> ::=

<record name>|<set name>|<realm name>|<char exp>

1. The <record name> phrase is used to indicate the name of a record which is found in the Sub-schema. The representation of the <record name> phrase must follow the rules for a well-formed FORTRAN symbolic name as defined in FORTRAN [1] Section 2.
2. The <set name> phrase is used to indicate the name of a set type which is found in the Sub-schema. The representation of the <set name> phrase must follow the rules for a well-formed FORTRAN symbolic name as defined

in FORTRAN [1] Section 2.

3. The <realm name> phrase is used to indicate the name of a realm type which is found in the Sub-schema. The representation of the <realm name> phrase must follow the rules for a well-formed FORTRAN symbolic name as defined in FORTRAN [1] Section 2.
4. The <char exp> phrase is used to represent the name of a record, set, or realm which is found in the Sub-schema but is either:
 - a. Not a well-formed FORTRAN symbolic name (See also Section 3.2 on the ALIAS statement), or
 - b. The name of an appropriately dimensioned CHARACTER variable which contains the desired record, set, or realm name during execution of the user's FORTRAN program (See CHARACTER variable as defined in FORTRAN [1] Section 8.4.2).
5. Note that name formation rules for record, set, realm and Sub-schema names are those used in the Schema for symbolic names, while rules for item names are those for FORTRAN symbolic names (compare with identifier in Section 2.3.11).
6. This report uses the <db name> phrase to identify entities which are manipulated by the Data Base Control System(DBCS). These entities are records, sets, and realms. They are not FORTRAN data structures in the sense that their names may be used in expressions or assignment statements. Record, set, and realm constructs are logical structures at increasingly abstract conceptual levels. The DBCS manipulates these entities in performing DML statements. The ultimate result of the DBCS action may be the acquisition or alteration of program variables contained in one or more data base records.

```
*****
*
* RECORD=<db name> phrase may be constructed as:
*
* RECORD=EMPNO      - a record name in the Sub-schema
* RECORD=AA2134     - a CHARACTER variable
* RECORD='EMP-NUMB' - literal name of a record
*                  in the Sub-schema
*
*****
```

2.3.7 Data Base Names

<db names> ::=

<db name> | (<db name> [, <db name>] ...)

1. More than one name may be supplied in certain of the

constructs allowed herein. An example is the FINISH command. The syntax shown previously indicates that a single name may be used anywhere the <db names> phrase is found, or a parenthesized list of names may be used wherever the <db names> phrase is legal. Thus the following examples are valid use of the <db names> phrase as shown in the FINISH syntax:

```
FINISH ( REALM='PAYROLL')
```

```
FINISH ( REALM=('PAYROLL',FICA,INSURE))
```

The use of ('PAYROLL', FICA,INSURE) is a valid construction of the <db names> phrase.

2. Individual implementations of DBCS may restrict the maximum number of the <db name> phrase occurrences within the <db names> phrase.

```
*****
*                                     *
*   REALM=ENG                       *
*   REALM=( ENG )                   *
*   REALM=( ENG, MFG )              *
*   REALM=( ENG, MFG, SHIP )        *
*   REALM=( 'ENG' , 'MFG' , SHIP! ) *
*   REALM=( VARXYZ , VARABC , SHIP ) *
*                                     *
*****
```

2.3.8 Data Base Procedure

```
<db proc> ::=
```

```
    <subroutine name>[( <argument> )]
```

2.3.9 END Phrase

```
<end> ::=
```

```
    END={ <statement number> | { <subroutine name> [( <argument> ) ] } }
```

The <end> phrase allows dynamic selection of the action to be taken when the DBCS encounters the end of a set or realm (status code 02100) during execution of a FETCH or FIND statement. The <end> phrase specifies, for the statement attempted, the statement number or the subroutine name (and arguments) to receive control.

1. If the <statement number> phrase is used, the indicated statement must appear in the same program unit as that containing the END=<statement number> clause; e.g. if the DML statement containing the <end> phrase is in the main program body, then the statement referenced by the <statement number> phrase must also occur in the main program body.

2. If the END=<subroutine name> clause is used, the indicated subroutine must be available to the FORTRAN program. If the routine invocation requires arguments, a parenthesized argument list may be used.
3. If an <end> phrase is used, the indicated program unit address receives control after the DBCS has completed initial cleanup actions. The special registers (See Section 5) are not cleared by the DBCS, but contain information related to the end-of-set data base status value causing the transfer of control to take place. Thus the <end> phrase control statement number or subroutine may utilize the special registers. Note that an <end> phrase will be executed instead of any USE statements in force.

```
*****
*
*      END=999
*      END=DBESUB
*      END=RECOVER ( 210, DBRELM, DBREC )
*
*****
```

2.3.10 Error Phrase

<error> ::=

ERR={<statement number>|{<subroutine name> [(<argument>)]}}

The <error> phrase allows dynamic selection of the action to be taken when the DBCS returns a non-zero status at the completion of a DML statement. The <error> phrase specifies, for the statement attempted, the statement number or the subroutine name (and arguments) to receive control.

1. If the <statement number> phrase is used, the indicated statement must appear in the same program unit as that containing the ERR=<statement number> clause; e.g. if the DML statement containing the <error> phrase is in the main program body, then the statement referenced by the <statement number> phrase must also occur in the main program body.
2. If the ERR=<subroutine name> clause is used, the indicated subroutine must be available to the FORTRAN program. If the routine invocation requires arguments, a parenthesized argument list may be used.

3. If an <error> phrase is used, the indicated program unit address receives control after the DBCS has completed initial cleanup actions. The special registers (See Section 5) are not cleared by the DBCS, but contain information related to the non-zero data base status value causing the transfer of control to take place. Thus, the <error> phrase control statement number or subroutine may utilize the special registers. Note that an <error> phrase will be executed instead of any USE statements in force.

```
*****
*
*      ERR=999
*      ERR=DBESUB
*      ERR=RECOVER ( 210, DBRELM, DBREC )
*
*****
```

2.3.11 Identifier

<identifier> ::=

as defined in FORTRAN [1] Section 2.5. This in turn refers to FORTRAN [1] section 2.2, where a symbolic name is defined.

2.3.12 Identifier Item

<id item> ::=

<identifier>

An <id item> phrase is a variable as defined in FORTRAN [1] Sections 2.2 and 2.5. The DBCS uses the value of the variable as a calculation key during set selection (See Section 3.4) when so directed by the Sub-schema.

2.3.13 Identifier List

<id list> ::=

<id spec> [, <id spec>] ...

2.3.14 Identifier Specification

<id spec> ::=

Input list item, as defined in 12.8.2.1 of FORTRAN [1], or implied DO list as defined in 12.8.2 of FORTRAN [1].

2.3.15 Integer Constant

<integer> ::=

as defined in FORTRAN [1] Section 4.3.1.

2.3.16 Item List

<item list> ::=

<item spec> [, <item spec>] ...

2.3.17 Item Specification

<item spec> ::=

Input list item as defined in FORTRAN [1] 12.8.2.1.

2.3.18 Realm Name

<realm name> ::=

the name of a realm in the Sub-schema. See the <db name> phrase in Section 2.3.6 and Section 3.3, The REALM Statement.

2.3.19 Record Name

<record name> ::=

the name of a Sub-schema record type. See the <db name> phrase in Section 2.3.6 and Section 3.5, The RECORD Statement.

2.3.20 Retaining Clause

<retain> ::=

<retain1> | <retain2>

If the user does not wish the data base currency indicators to be changed by the action of a DML statement, the <retain> phrase must be used with that statement. The statements which can have a retaining clause are FETCH, FIND and STORE.

FORMAT 1. Retaining Individual Currency Indicators.

<retain1> ::=

RETAINING=([RECORD][,REALM][,SET|{SET=<db names>}}])

FORMAT 2. Retaining all Currency indicators

<retain2> ::=

RETAINING= MULTIPLE

1. If more than one keyword clause is used in a <retain> phrase, then the keyword is enclosed in a parenthesized list. If the SET=<db names> option is used as the sole clause in the <retain> phrase, then it must be enclosed in parentheses, e.g. see the following examples.
2. If the keyword REALM is used, no realm currency indicators are updated.
3. If the keyword SET is used without specifying the <db names> phrase, no set type currency indicators are updated.
4. If the keyword RECORD is used, no record type currency indicators are updated.
5. If the SET=<db name> clause is used, the currency indicators for the indicated set(s) are not updated.
6. If the keyword MULTIPLE is used, no realm, set or record type currency indicators are updated.
7. The retaining clause allows the user to keep certain data base currency indicators in their previous state while executing DML statements which would otherwise update these currency indicators.

```

*****
*
*   RETAINING=MULTIPLE
*   RETAINING=RECORD
*   RETAINING=REALM
*   RETAINING=( RECORD )
*   RETAINING=( RECORD, REALM )
*   RETAINING=( SET=EXEMPT )
*   RETAINING=( SET=( HOURLY , EXEMPT ) )
*   RETAINING=( RECORD, SET=( HOURLY , EXEMPT ) )
*   RETAINING=( RECORD, REALM, SET=( HOURLY , EXEMPT ) )
*
*****

```


2.3.21 Schema Item Name

<schema item name> ::=

the name of a data item in the Schema.

2.3.22 Schema Name

<schema name> ::=

the name of an object Schema. This name follows the rule for an <identifier> phrase.

2.3.23 Set Name

<set name> ::=

the name of a Sub-schema set type. See the <db name> phrase in Section 2.3.6 and Section 3.4 on the SET statement.

2.3.24 Statement Number

<statement number> ::=

as defined in FORTRAN [1] Section 3.4.

2.3.25 Status List

<status list> ::=

a sequence of character constants enclosed in quotes and separated by commas. See Section 4.15.

2.3.26 Subroutine Name

<subroutine name> ::=

as defined in FORTRAN [1] Section 18.2.3.

2.3.27 Sub-schema Name

<subschema name> ::=

the name of an object Sub-schema. This name follows the rules for an <identifier> phrase.

2.3.28 Usage

<usage> ::=

[PROTECTED|EXCLUSIVE|CONCURRENT][, RETRIEVAL|UPDATE]

1. The PROTECTED option prevents any other run unit from updating affected realms. Other run units may access a realm readied in this mode.

2. The EXCLUSIVE option prevents any other run unit from executing a READY statement for the affected realms.
3. The CONCURRENT option allows other run units to ready the affected realms for RETRIEVAL or CONCURRENT update.
4. The RETRIEVAL option permits only access to the affected realms.
5. The UPDATE option permits both access to, and modification of, the contents of the affected realms.

Withdrawn/Retirée

2.4 RECORD SELECTION EXPRESSIONS

Record selection expressions are used to specify the criteria whereby the DBCS is to select a record in the data base. The record thus selected becomes the current record of the run unit, upon which subsequent statements may operate when accessing the data base. A record selection expression is used only in the FIND and FETCH statements. If a currency indicator is used, either a current record or a record relative to a current record is selected.

<rse> ::=

<rse 1>|<rse 2>|<rse 3>|<rse 4>|<rse 5>|<rse 6>|<rse 7>

There are seven formats of the record selection expression. These formats permit both relative and absolute selection criteria. Records may be selected according to their contents, data base key values, or by currency indicators. The various record selection expressions are used as follows:

Format 1 - <rse 1> Direct Access

Format 2 - <rse 2> Calculate Mode Access

Format 3 - <rse 3> Set Search Access

Format 4 - <rse 4> Positional Access

Format 5 - <rse 5> Currency Indicator Access

Format 6 - <rse 6> Set Owner Access

Format 7 - <rse 7> Set Occurrence Selection Rules Access

2.4.1 Format 1 - Direct Access

<rse 1> ::=

KEY=<db key> [, RECORD=<db name>]

1. The name supplied as the <db key> phrase must reference an integer variable or function as defined previously.
2. The record name phrase is optional. If used, it acts as a check in order to ensure that the object record of the statement has the indicated record type. If the object record is not of the indicated type, an exception condition occurs (See Section 5, Exception Handling).

2.4.2 Format 2 - Calculate Mode Access`<rse 2> ::=``{DUPLICATE|ANY} , RECORD=<db name>`

The record referenced by the <db name> phrase must have a calculation location mode.

1. When the ANY phrase is used, the calculation key values in the record area referenced by the <db name> phrase are used to calculate the data base key value of the identified record. If more than one record in the data base contains calculation key values equal to those in the associated record area, the record identified is the one whose data base key has the lowest value.
2. The processing of the DUPLICATE phrase is the same as for the ANY phrase unless the calculation values of the record identified are all equal to the calculation key in the current record of the run unit (obeying the rules for comparison of operands in a relation condition). The DBCS searches for the identified record beginning with the current record of the run unit, and proceeds in sequence by ascending data base key values. The first record found after the current record of the run unit (with the proper calculation key values) is the record identified.

2.4.3 Format 3 - Set Search Access`<rse 3> ::=``DUPLICATE, SET=<db name> , USING=<id list>`

The identifiers supplied by the <id list> phrase specified in the USING clause must be defined in the Sub-schema record description entry of the current record of the set type indicated by the <db name> phrase in the SET clause.

The DBCS selects the member of the set type referenced in the SET=<db name> clause whenever:

1. The record type is equal to that of the current record of the set named by the SET=<db name> clause, and
2. The contents of the data items (variables) referenced in the USING clause are equal respectively to all those in the current record of the set named in the SET=<db name> clause (obeying the rules for comparison of operands in a relational condition).

The DBCS begins its search for the selected record at the current record of the specified set type, and then proceeds to search in the order defined by the set-ordering criteria for that set type.

2.4.4 Format 4 - Positional Access

<rse 4> ::=

<posit>[,RECORD=<db name>][,{REALM|SET}=<db name>]

<posit> ::=

{OFFSET=<int exp>}|FIRST|LAST|NEXT|PRIOR

<int exp> ::=

an integer expression (See FORTRAN [1] 6.1) which indicates the ordinal position of a record within the realm selected by the REALM=<db name> clause, or the current realm of run-unit if the REALM=<db name> clause is not used. Explicitly, offset is measured from the initial record occurrence of the set; i.e., zero means the owner.

1. If the RECORD=<db name> clause is not used, all record types specified in the Sub-schema within the set referenced by SET=<db name> clauses, or the realm referenced by the REALM=<db name> clause, are considered in evaluating the record selection expression.
2. If the RECORD=<db name> clause is used, only the indicated record type is considered in evaluating the record selection. All other record types are ignored.
3. If the REALM=<db name> clause is used, only a record stored within the indicated realm is identified.
4. If the SET=<db name> clause is used, only a record that is a member of the indicated set is identified.
5. If both the NEXT and REALM=<db name> clauses are used, the record identified is that whose data base key value is the next higher, relative to the value of the data base key value of the current record in the indicated realm.
6. If both the PRIOR and REALM=<db name> clauses are used, the record identified is that whose data base key value is the next lower, relative to the value of the data base key value of the current record in the indicated realm.
7. If both the FIRST and REALM=<db name> clauses are used, the record identified is that whose data base key value is the lowest, relative to all records stored in the indicated realm.
8. If both the LAST and REALM=<db name> clauses are used, the record identified is that whose data base key value is the highest, relative to all records stored in the indicated realm.

9. If both the NEXT and the SET=<db name> clauses are used, the record identified is the next member record of the set, relative to the current member of the indicated set, according to the ordering criteria for that set type.
10. If both the PRIOR and the SET=<db name> clauses are used, the record identified is the prior member record of the set, relative to the current member of the indicated set, according to the ordering criteria for that set type.
11. If both the FIRST and the SET=<db name> clauses are used, the record identified is the first member of the set in which the current record of that set type is a tenant, according to the set ordering criteria for the indicated set type.
12. If both the OFFSET=<int exp> and the SET=<db name> clauses are used, the record identified is that whose ordinal position in the set in which the current record indicated (by the SET=<db name> clause) is a tenant, is equal to the value of the <int exp> phrase. If the value of the <int exp> phrase is positive, the ordinal position is relative to the first member of that set, according to the set ordering criteria for that set type. If negative, the ordinal position is relative to the last member of that set, according to the set ordering criteria for that set.

2.4.5 Format 5 - Currency Indicator Access

<rse 5> ::=

CURRENT[,RECORD=<db name>][, {REALM|SET}=<db name>]

1. If both the RECORD=<db name> and the SET=<db name> clauses appear, the indicated record must be the current record of the set type named by the SET=<db name> clause.
2. If both the RECORD=<db name> and the REALM=<db name> clauses appear, the indicated record must be the current record of the realm named by the REALM=<db name> clause.
3. If only the RECORD=<db name> clause is used, the record identified is the current record of the record type identified by the <db name> phrase.
4. If only the SET=<db name> clause is used, the record identified is the current record of the set type identified by the <db name> phrase.
5. If none of the optional clauses appear, the record indicated is the current record of the run unit.

2.4.6 Format 6 - Set Owner Access

<rse 6> ::=

OWNER, SET=<db name>

1. The set type referenced by the SET=<db name> clause must not be a singular set.
2. The record identified is the owner of the set in which the current record name of the SET=<db name> clause is a tenant.

2.4.7 Format 7 - Set Occurrence Selection Rules Access

<rse 7> ::=

RECORD=<db name>, [CURRENT] SET=<db name> [, USING=<id list>]

1. The record identified has a record type equal to that referenced by the RECORD=<db name> clause. The record identified is a member of the set named in the SET=<db name> clause.
2. If the CURRENT keyword is used, the record identified is a member of the set in which the current member of the set named by the SET=<db name> clause is a tenant.
3. If the CURRENT keyword is not used, the record identified is a member of the set specified by the set selection criteria of the set named by the SET=<db name> clause.
4. If the USING=<id list> clause is used, the contents of the data items named (within the identified record) are all equal to the contents of the data items, respectively, in the record named by the RECORD=<db name> clause, according to the rules for comparison of operands in a relational condition.
5. If the USING=<id list> clause is not used, the record identified is the first member of the indicated set, according to the set ordering criteria for that set type.
6. The DBCS begins its search at the first member record and proceeds in its search for the identified record according to the set ordering criteria for that set type. The record identified is the first one meeting the previous requirements.


```

*****
*
*   RECORD SELECTION EXPRESSIONS EXAMPLES
*
*   Format 1.      (KEY=DBK, RECORD=RECA)
*
*   Format 2.      (ANY,          RECORD=RECA)
*                  (DUPLICATE, RECORD=RECA)
*
*   Format 3.      (DUPLICATE, SET=DEPT1, USING=(A,B,C))
*
*   Format 4.      (OFFSET=I      , RECORD=RECA, SET=DEPT1)
*                  (OFFSET=-5    , RECORD=RECA, REALM=EMPFIL)
*                  ( FIRST      , RECORD=RECA, SET=DEPT1)
*                  ( LAST       , RECORD=RECA, SET=DEPT1)
*                  ( NEXT       , RECORD=RECA, SET=DEPT1)
*                  ( PRIOR      , RECORD=RECA, SET=DEPT1)
*
*   Format 5.      (CURRENT, RECORD=RECA, SET=DEPT1)
*                  (CURRENT,          REALM=EMPFIL)
*                  (CURRENT)
*
*   Format 6.      (OWNER, SET=DEPT1)
*
*   Format 7.      (RECORD=RECA, CURRENT SET=DEPT1, USING=(A,B))
*                  (RECORD=RECA,          SET=DEPT1, USING=ABCDE)
*
*****

```


2.5 TENANCY CONDITIONS

There are three tenancy conditions; each returns a logical result value, indicating whether the current record of the run unit is a member, owner, or tenant of the indicated set.

<tenancy condition> ::=

{MEMBER(<db name>)} | {OWNER(<db name>)} | {TENANT(<db name>)}

1. Only the set named by the <db name> phrase is considered in determining the logical result value of the condition.
2. If MEMBER is used, the result of the condition is .TRUE. if the current record of the run unit is a member of the indicated set.
3. If OWNER is used, the result of the condition is .TRUE. if the current record of the run unit is the owner of the indicated set.
4. If TENANT is used, the result of the condition is .TRUE. if the current record of the run unit is a member or an owner of the indicated set.
5. If an argument is invalid, the result of the condition is .FALSE.

```
*****
*
*      IF(      MEMBER(EXEMPT)) GO TO 101      *
*      IF(.NOT. MEMBER(EXEMPT)) GO TO 102      *
*      IF(      OWNER (EXEMPT)) GO TO 103      *
*      IF(.NOT. OWNER (EXEMPT)) GO TO 104      *
*      IF(      TENANT(EXEMPT)) GO TO 105      *
*      IF(.NOT. TENANT(EXEMPT)) GO TO 106      *
*
*****
```


2.6 MEMBERSHIP CONDITIONS

There are three membership conditions; each returns either .TRUE. or .FALSE., indicating whether the current record of the run-unit is first or last of SET, or whether the current set occurrence is empty.

<member condition> ::=

{EMPTY(<db name>)} | {FIRST(<db name>)} | {LAST(<db name>)}

1. If the <db name> phrase does not reference a singular EMPTY set, the object set occurrence is the one specified by the current record of the set type referenced by the <db name> phrase. If the <db name> phrase references a singular set, the object set is that singular set. If the object set has no members, the result of the EMPTY test is .TRUE.
2. If the FIRST condition is used, the result will be .TRUE. if the current record of run-unit is the first member occurrence of the set identified by the <db name> phrase.
3. If the object set has members and the LAST test is used, the result will be .TRUE. if the current member of that set is the last occurrence in that set.
4. If the <db name> phrase is not a valid set-name or (for FIRST and LAST) there is no currency of run-unit, the result of the condition is .FALSE.

```
*****
*
*      IF(      EMPTY(EXEMPT)) GO TO 101
*      IF(.NOT. EMPTY(EXEMPT)) GO TO 102
*      IF(      FIRST(EXEMPT)) GO TO 103
*      IF(.NOT. FIRST(EXEMPT)) GO TO 104
*      IF(      LAST (EXEMPT)) GO TO 105
*      IF(.NOT. LAST (EXEMPT)) GO TO 106
*
*****
```


2.7 NULLITY CONDITION

The nullity condition determines whether a specified item has been assigned a value.

<null cond> ::=

NULL (<identifier>)

1. The <null cond> phrase determines whether a specified item has been assigned a value.
2. The <identifier> phrase is a variable which must appear in the Sub-schema. If the data item (variable) has a null attribute (no value), the result of the nullity condition is .TRUE. Otherwise the result is .FALSE. If the variable name references a group item (e.g. a record name), the result is .TRUE. only if every item within that group has the null data attribute.
3. If an argument is invalid, the result of the condition is .FALSE.

```
*****
*
*          IF(      NULL(TAXES)) GO TO 101
*          IF(.NOT. NULL(TAXES)) GO TO 102
*
*****
```


3. FORTRAN SUB-SCHEMA

A Sub-schema for use by FORTRAN consists of an initial SUBSCHEMA statement, followed by a set of ALIAS, REALM and SET statements, followed by a set of RECORD and type-statements and terminated with an END statement. It should be noted that a Sub-schema program unit may not necessarily be compiled by a FORTRAN compiler, but its converted form must be used by the compiler to resolve references.

A FORTRAN Sub-schema must contain at least one REALM statement.

Privacy locks established in the Sub-schema are logically appended to locks which are specified in the Schema, i.e., Sub-schema locks do not override Schema-level locks.

The syntax of the SUBSCHEMA, ALIAS, REALM, SET, RECORD, and type-statements are described in the following sections.

The required order of statements and comment lines in the Sub-schema is illustrated in the table below. Note that those type-statements which immediately follow a RECORD statement are part of that record.

*	*	*
*	SUBSCHEMA statement	*
*	*	*
*	*****	*
*	*	*
*	ALIAS statements	*
* Comment	*	*
*	*****	*
*	*	*
* Lines	REALM and SET statements	*
*	*	*
*	*****	*
*	*	*
*	RECORD and type-statements	*
*	*	*

*		*
*	END statement	*
*		*

3.1 SUBSCHEMA STATEMENT

FUNCTION

The SUBSCHEMA statement specifies the name of the Sub-schema, identifies the subject Schema to which the Sub-schema applies, can specify privacy lock values associated with the use of the Sub-schema, and can also specify data base procedure name(s) to be called when the Sub-schema is invoked or finished.

FORMAT

SUBSCHEMA {<char con>|<schema name>} ,SCHEMA={<char con>|<schema name>
[,<subschemata control>] ... [,<subschemata lock option>] ...
[,<subschemata key option>]}

<subschemata control> ::=

CALL <db proc> {BEFORE|AFTER|ERROR} {INVOKE|FINISH}

<subschemata lock option> ::=

LOCK(<lock list>)=<lock spec>

<lock list> ::=

ALL | {<subschemata lock type>[,<subschemata lock type>]... }

<subschemata lock type> ::=

INVOKING | ALTERING | LOCKS

<lock spec> ::=

<char con>|<db proc>

<subschemata key option> ::=

KEY=<lock spec>

RULES

1. The Sub-schema is identified by either a <subschemata name> or a <char con> phrase as specified below:
 - a) If the <subschemata name> phrase is used, the representation of <subschemata name> phrase identifies the Sub-schema to the DBCS,
 - b) If the <char con> phrase is used, the value of the <char con> phrase is used to identify the Sub-schema to the DBCS.

2. The <subschema control> phrase can be used to identify a data base procedure which is to be called when an INVOKE and/or a FINISH statement is executed. The data base procedure named by the <db proc> phrase will be called according to the following rules:
 - a) If BEFORE is used, the indicated data base procedure will be called by the DBCS prior to execution of the associated FORTRAN statement,
 - b) If AFTER is used, the indicated data base procedure will be called by the DBCS subsequent to the execution of the associated FORTRAN statement,
 - c) If ERROR is used, the data base procedure will be called by the DBCS only if an error is detected during execution of the indicated FORTRAN statement as described in Section 5.2
3. The <subschema key option> phrase is used during the process of placing the Sub-schema in a Sub-schema library. If, and only if, the KEY=<lock spec> clause satisfies the privacy lock (if any) associated with the process, the Sub-schema is placed in the Sub-schema library. The rules for forming the <lock spec> phrase are as follows:
 - a) If the <char con> phrase is used, the value of the character constant represents the privacy key,
 - b) If the <db proc> phrase is used, acceptance of the privacy key will depend upon the execution of the identified data base procedure.
4. The <subschema lock option> phrase is used to specify privacy locks associated with the use of the Sub-schema. The locks so specified must be satisfied by a PRIVACY statement in a FORTRAN program.
5. A <lock list> phrase identifies individual privacy locks which must be satisfied by a PRIVACY statement.
6. If INVOKING is used, the indicated <lock spec> phrase will be used when validating invocation of the Sub-schema by an INVOKE statement in a FORTRAN program.
7. If ALTERING is used, the indicated <lock spec> phrase will be used when validating modifications to the Sub-schema.
8. If LOCKS is used, the indicated <lock spec> phrase will be used when validating authorization to change any privacy lock or privacy key phrase in the Sub-schema.
9. If ALL is used, the indicated <lock spec> phrase will be used as though INVOKING, ALTERING and LOCKS were specified.

10. If no <lock list> phrase is present, the <subschema lock option> phrase is given a <lock list> phrase of INVOKING.

```

*****
*
*   FORTRAN:
*
*   SUBSCHEMA SUB1, SCHEMA='DATA-BASE'
*   SUBSCHEMA SUB1, SCHEMA='DATA-BASE', KEY='DOIT'
*   SUBSCHEMA SUB1, SCHEMA='DATA-BASE',
*       CALL UMONTR BEFORE INVOKE,
*       CALL UCLOSE AFTER FINISH,
*       LOCK(INVOKING, LOCKS)='LOCK2'
*       LOCK(ALTERING)='OK TO ALTER'
*
*   COBOL:
*
*   SS SUB1 WITHIN DATA-BASE.
*   SS SUB1 WITHIN DATA-BASE
*       PRIVACY KEY IS "DOIT".
*   SS SUB1 WITHIN DATA-BASE
*       PRIVACY LOCK FOR INVOKING, LOCKS IS "LOCK2"
*       PRIVACY LOCK FOR ALTERING IS "OK TO ALTER".
*
*****

```


3.2 ALIAS STATEMENT

FUNCTION

The ALIAS statement permits a name defined in the Schema to be renamed within the FORTRAN Sub-schema. It is envisioned that the ALIAS statement will be used to provide FORTRAN-acceptable names for COBOL(Schema) named entities, specifically data item names which the programmer references as conventional FORTRAN variables.

FORMAT

ALIAS [(<schema name type>)] <re-name> [,<re-name>]...

<schema name type> ::=

REALM|SET|RECORD|ITEM

<re-name> ::=

{<char con>|<identifier>} = <name in schema>

<name in schema> ::=

<char con>

The value of the <name in schema> phrase has the following syntax:

<realm name>|<set name>|<schema item name>|<record name>|
<record name>.<schema item name>

RULES

1. A <name in schema> phrase which does not have the form <record name>.<schema item name> is called an unqualified reference. A <name in schema> phrase which has the form <record name>.<schema item name> is called a qualified reference.
2. If the <name in schema> phrase is a qualified reference then the <schema name type> phrase, if present, must be an (ITEM).
3. If a (REALM) is present, the search group consists of the subject Schema realm names.
4. If a (SET) is present, the search group consists of the subject Schema set names.
5. If a (RECORD) is present, the search group consists of the subject Schema record names.
6. If an (ITEM) is present and the <name in schema> phrase is an unqualified reference, the search group consists of the subject Schema data item names.

7. If an (ITEM) is present and the <name in schema> phrase is a qualified reference, the search group consists of those subject Schema data item names which are contained within the subject Schema record identified by the <char con> or <identifier> phrase.
8. The <name in schema> phrase must be equal to exactly one of the names in the search group.
9. If the <schema name type> phrase is absent and the <name in schema> phrase is unqualified, the search group consists of all the realm, set, record, and item names in the subject schema.
10. The <char con> or <identifier> phrase in the syntax becomes the alias of the <name in schema> phrase and is the name that must be referenced by the FORTRAN program.
11. A given <char con> or <identifier> phrase may appear only once in the ALIAS statements in the Sub-schema.
12. A given <name in schema> phrase may appear only once in the ALIAS statements in the Sub-schema.

Note that the <char con> or <identifier> phrase syntax construct allows either literals representing COBOL-like names, or an identifier, to be used when referring to REALM, SET, or RECORD names. In the latter case, an identifier must be formed according to the rules for a FORTRAN identifier. If the Schema form of a name does not conform to the FORTRAN identifier rules, a choice can be made to require either that a literal be used for the name, or that an ALIAS statement be provided in the Sub-schema to re-name the COBOL representation in a form suitable to the FORTRAN program.

```

*****
*
*  FORTRAN:
*
*  ALIAS FIELD1='FIELD-1'
*  ALIAS DATE2='REC1.DATE'
*  ALIAS(RECORD) PRIME='GROUP'
*  ALIAS AMT='PAY-AMOUNT',TAX='LOCAL-TAX'
*  ALIAS 'P-AMT' = 'PAY-AMOUNT'
*
*  COBOL:
*
*  MAPPING DIVISION.
*  ALIAS SECTION.
*      AD ==FIELD-1== BECOMES FIELD1.
*      AD ==REC1;DATE== BECOMES DATE2.
*      AD ==GROUP== RECORD-NAME BECOMES PRIME.
*      AD ==LOCAL-TAX==BECOMES TAX.
*
*****

```


3.3 REALM STATEMENT

FUNCTION

The REALM statement specifies the realms that are to be made available to a FORTRAN Sub-schema. It can also specify privacy lock values to be associated with the use of the realm and data base procedure(s) to be called when accessing the realm.

FORMAT

REALM<realm declaration>[,<realm declaration>]...

<realm declaration> ::=

<db name> [<realm control>] ... [(<realm lock list>)]

<realm control> ::=

CALL <db proc> {BEFORE | AFTER | ERROR} <realm lock>

<realm lock> ::=

ALL | {[EXCLUSIVE|PROTECTED]{RETRIEVAL | UPDATE}}

<realm lock list> ::=

<realm lock option>[,<realm lock option>]...

<realm lock option> ::=

LOCK<realm lock type>=<lock spec>

<realm lock type> ::=

(<realm lock>[,<realm lock>]...)

<lock spec> ::=

<char con>|<db proc>

RULES

1. A declaration is established for each <db name> phrase. This declared name may appear in a syntactic context which requires a <realm name> phrase (See Section 2.3.6).
2. If the <realm control> phrase is used, the data base procedure identified by the <db proc> phrase will be called by the DBCS when the realm identified by the <db name> phrase is readied by a FORTRAN READY statement. See Section 3.1, rules 2a, 2b, and 2c for a description of the action of the keywords BEFORE, AFTER and ERROR.

3. A <realm lock> phrase can have one of seven possible constructions: EXCLUSIVE RETRIEVAL, EXCLUSIVE UPDATE, PROTECTED RETRIEVAL, PROTECTED UPDATE, RETRIEVAL, UPDATE or ALL. If ALL is used, the action of the <realm lock> phrase is as though EXCLUSIVE RETRIEVAL, EXCLUSIVE UPDATE, PROTECTED RETRIEVAL, PROTECTED UPDATE, RETRIEVAL and UPDATE were specified.
4. If no <realm lock type> phrase is present, the <realm lock option> phrase acts as though a <realm lock> phrase of RETRIEVAL were specified.

```

*****
*
*   FORTRAN:
*
*       REALM REALM1,REALM2
*       REALM REALM1
*           CALL PROC1 BEFORE EXCLUSIVE UPDATE (LOCK='LOCK1'),
*           REALM2(LOCK(EXCLUSIVE UPDATE)='EU',
*           LOCK(RETRIEVAL)='R')
*
*   COBOL:
*
*       REALM SECTION.
*           RD REALM1.
*           RD REALM2.
*
*       REALM SECTION.
*           RD REALM1 PRIVACY LOCK IS "LOCK1".
*           RD REALM2 PRIVACY LOCK FOR EXCLUSIVE UPDATE IS "EU"
*           PRIVACY LOCK FOR RETRIEVAL IS "R".
*
*****

```


3.4 SET STATEMENTFUNCTION

The SET statement specifies the sets that are to be made available to a FORTRAN Sub-schema with provision being made to specify one or more selection algorithms for a set occurrence. It can also specify privacy lock values to be associated with the use of the set and data base procedures to be called when using sets.

FORMAT

SET <set declaration> [,<set declaration>]...

<set declaration> ::=

<db name>[(<set option> [,<set option>]...)]

<set option> ::=

<set control>|<selection option>|<set lock option>

<set control> ::=

CALL <db proc> {BEFORE | AFTER | ERROR}
{ {INSERT|REMOVE}| {FIND|FETCH}
[OWNER | NEXT | FIRST | LAST | PRIOR | VIASET | OFFSET]}

<selection option> ::=

SELECTION({RECORD=<db name> ,ENTRY <db name>=<entry specifier>
[,<path specifier>] ...} | DBPROC=<db proc>)

<entry specifier> ::=

SYSTEM|CURRENT|<calc entry>|DIRECT(<db key>)

<calc entry> ::=

CALC<key list>

<key list> ::=

(<key item>[,<key item>] ...)

<key item> ::=

<id item>[=<identifier>]

<path specifier> ::=

PATH = (<db name>[,<key item>]...)

<set lock option> ::=

LOCK<set lock list>=<lock spec>


```

<set lock list> ::=

    ALL | {( <set lock type> [, <set lock type> ] ... ) }

<set lock type> ::=

    CONNECT | DISCONNECT | FIND | ORDER | MODIFY | FETCH

<lock spec> ::=

    <char con> | <db proc>

```

RULES

1. Each <set option> phrase has an object set which is defined to be the subject Schema set referred to by the set <db name> phrase which immediately precedes that <set option> phrase.
2. Each <selection option> phrase has an entry set which is the subject Schema set referred to by the ENTRY phrase. If no entry set is used, a data base procedure is used to specify the entry set dynamically during execution of the FORTRAN user's program.
3. Each <path specifier> phrase has a path set which is the subject Schema set referred to by the <db name> phrase of that <path specifier> phrase.
4. No two record <db name> phrases in the same <set option> phrase may refer to the same subject Schema record.
5. The subject Schema record referred to by a record <db name> phrase must be defined as a member record of the object set.
6. If a <selection option> phrase has no <path specifier> phrases, the ENTRY phrase must refer to the object set.
7. If a <selection option> phrase has any <path specifier> phrases, both of the next two constraints apply:
8. The path set of the leftmost <path specifier> phrase must have an owner record which is a member record of the entry set.
9. The path set of the rightmost <path specifier> phrase must be the object set.
10. If a <selection option> phrase has more than one <path specifier> phrase, then the following constraint must be true for each adjacent pair of <path specifier> phrases: The path set of the right <path specifier> phrase of that pair must have an owner record which is a member record of the path set of the left <path specifier> phrase of that pair.

11. The SYSTEM phrase may be present only if the entry set is a singular set.
12. The DIRECT phrase may not be present if the entry set is a singular set.
13. The <calc entry> phrase may be present only if the owner record of the entry set has a Schema defined LOCATION MODE IS CALC...DUPLICATES NOT ALLOWED clause.
14. If a <db key> phrase is a <variable name> phrase, that <variable name> phrase must have an integer data type.
15. An <id item> phrase which appears in an <entry specifier> phrase must be contained within a subject Schema record which is the owner of the entry set.
16. An <id item> phrase which appears in a <path specifier> phrase must be contained within a subject Schema record which is the owner of the path set.
17. Each <id item> phrase in a <calc entry> phrase must refer to a calc key defined in the Schema definition of the owner record of the entry set.
18. The number of <id item> phrases in the <calc entry> phrase must equal the number of calc keys defined in the schema description of the owner record of the entry set.
19. The <key item> phrases in the first <path specifier> phrase must be defined in the Schema in such a way that the aggregate User Working Area value of these <key item> phrases cannot be equal to the aggregate data base value of those <key item> phrases in more than one occurrence of the owner record of the path set within the selected occurrence of the entry set.
20. The <key item> phrases in the second and subsequent <path specifier> phrases must be defined in the Schema in such a way that the aggregate User Working Area value of those <key item> phrases cannot be equal to the aggregate data base value of those <key item> phrases in more than one occurrence of the owner record of the path set within the selected occurrence of the immediately preceding path set.
21. A declaration is established for each set type identified as a <db name> phrase. This declared name may appear in a syntactic context which requires a <db name> phrase. If a <db key> phrase is an <identifier> phrase, a declaration is established for that <identifier> phrase. This declared name may appear in a syntactic context which requires a variable name. The declared name is given an integer data type.

variable name. The declared name is given the data type and characteristics of the <id item>. During set selection the contents of <identifier> are implicitly transferred to <id item> for use as a calculation key value by the DBCS.

23. During the run-time execution of a FIND or FETCH statement, the <selection option> phrase causes the set selection criteria, specified in the Schema for the subject schema record referenced by the <db name> phrase (See Section 2.4) as a member of the object set, to be replaced by the set selection criteria specified in the <selection option> phrase. The <selection option> phrase has no effect on other DML statements. A <variable lock list> phrase is used to specify individual privacy locks in the Sub-schema which must be satisfied by corresponding values in PRIVACY statements.
24. If a <set control> phrase is used, the data base procedure identified by the <db proc> phrase will be called by the DBCS when the set identified by the <db name> phrase in the <set declaration> phrase is accessed by the indicated DML statement type(s). See Section 3.1, rules 2a, 2b and 2c for a description of the action of the keywords BEFORE, AFTER, and ERROR.
25. A <set lock option> phrase is used to specify individual PRIVACY locks in the Sub-schema which must be satisfied by a corresponding value in a FORTRAN PRIVACY statement. Once a given privacy lock has been satisfied, the corresponding FORTRAN DML statement type may be executed.
26. If the keyword ALL is used in a <set lock list> phrase, the resultant action is as though a <set lock list> phrase of CONNECT, DISCONNECT, FETCH, FIND, ORDER, and MODIFY were used.
27. If no <set lock list> phrase is used, the <set lock option> phrase acts as though a <set lock list> phrase of FIND were specified.


```

*****
*
*   FORTRAN:
*
*   SET SET1,SET2,SET3
*   SET SET1(LOCK(CONNECT,FIND)='CF-LOCK')
*   SET SET1(SELECTION(RECORD=REC1,
*       PATH SET2(S2=S3,S4=S5),PATH SET3(S6=S7)))
*
*
*   COBOL:
*
*   SET SECTION.
*       SD SET1.
*       SD SET2.
*       SD SET3.
*
*   SET SECTION.
*       SD SET1 PRIVACY LOCK FOR CONNECT,
*       FIND IS "CF-LOCK".
*
*   SET SECTION.
*       SD SET3
*           SET SELECTION FOR REC1 IS
*           VIA SET1 OWNER DBK1
*           VIA SET2 OWNER VALUE OF S2 IS S3 S4 IS S5
*           VIA SET3 OWNER VALUE OF S6 IS S7.
*
*****

```


3.5 RECORD STATEMENT

FUNCTION

The RECORD statement specifies the records that are to be made available to a FORTRAN Sub-schema. By virtue of its positional relationship with type-statements within a data definition group, it identifies a collection of variable names whose values are related. It provides the ability to restrict the instances of records accessed by realm. It can also specify privacy lock values to be associated with the use of the record or procedures to be invoked.

FORMAT

RECORD <record declaration>[,<record declaration>] ...

<record declaration> ::=

<db name>[(<record option> [,<record option>] ...)]

<record option> ::=

<record control>|<within option>|<realm-id option>|
<record lock option>|<check option>

<record control> ::=

CALL <db proc> { BEFORE | AFTER | ERROR } <record lock list>

<record lock list> ::=

ALL | { <record lock type> [, <record lock type>] ... }

<record lock type> ::=

CONNECT|DISCONNECT|STORE|DELETE|MODIFY|FIND|GET|FETCH|
{ ERASE [ALL|PERMANENT|SELECTIVE] }

<within option> ::=

WITHIN<db names>

<realm-id option> ::=

REALMID=<char var>

<record lock option> ::=

LOCK(<record lock list>)=<lock spec>

<lock spec> ::=

<char con>|<db proc>

<check option> ::=

CHECK={<db proc> | DATA TYPE}

RULES

1. The subject record is the subject Schema record referred to by the <db name> phrase.
2. If the REALMID=<char var> clause is used, the indicated identifier must be a FORTRAN variable of CHARACTER type which has been declared appropriately to contain the characters which comprise the longest realm name in which the subject record could be stored via DML.
3. A <record option> phrase list may contain only one <within option> phrase.
4. Each <within option> phrase must refer to one of the subject Schema realms in which the Schema definition permits the subject record to be located.
5. If the <char var> phrase in a <realm-id option> phrase has not been explicitly declared as a Sub-schema data item or a FORTRAN variable, then a declaration will be established for that variable name during the INVOKE. The declared name is given a character data type whose size is sufficient to contain the characters which comprise the name of the longest realm name.
6. For each RECORD statement, one or more subject schema realms which constitute the range of interest of the FORTRAN Sub-schema are defined. If no <within option> phrase is present, the range of interest is all of the subject Schema realms in which the Schema definition permits the subject record to be located. If a <within option> phrase is present, the range of interest is those subject Schema realms referred to by the <db names> phrases. Within a DML program which invokes the FORTRAN Sub-schema, only those occurrences of records which are located within the range of interest are accessed.
7. When the Schema description permits a subject record to be located in more than one realm, a process called realm selection must be performed at the time an occurrence of that record is stored in the data base. Realm selection is described by the following algorithm.
 - a. Neither the <realm-id option> phrase nor the <within option> phrase is used in the Sub-schema RECORD statement.
 - (1) If the subject record cannot occur within any realm named by the Sub-schema REALM statements, then the FORTRAN DML program is in error.
 - (2) If the subject record can occur in one or more of the realms named in the Sub-schema REALM statements, and if the Schema does not specify a

method for realm selection, then the DBCS uses an implementor defined method to select an appropriate realm. The selected realm will be a realm which is named in a Sub-schema REALM statement.

- (3) If the subject record can occur in one or more realms, and if the Schema does specify a method for realm selection, then that method is used. If the selected realm is not referenced by a REALM statement in the Sub-schema, then the FORTRAN DML program is in error.
- b. The REALMID=<char var> clause is used, but the <within option> phrase is not used. Note that a <char var> phrase can reference a FORTRAN CHARACTER variable which contains the representation of a realm name during execution of the FORTRAN DML program.
- (1) If the <char var> phrase does not contain a realm name(e.g., it contains blank characters), then a realm is selected as described by rule 8a in the previous paragraphs.
 - (2) If the <char var> phrase contains a realm name, then
 - (a) If none of the REALM statements in the Sub-schema reference a realm in which the subject record can occur, then the FORTRAN DML is in error.
 - (b) If the Schema does not specify a method of realm selection, and if the contents of the <char var> phrase represent the name of a realm in which the record can occur, and if that realm is referenced by a Sub-schema REALM statement, then that realm is selected, otherwise the FORTRAN DML program is in error.
 - (c) If the Schema does specify a method of realm selection, then that realm is chosen. If the contents of the <char var> phrase do not represent the name of the selected realm, then the FORTRAN DML program is in error.
- c. The <within option> phrase is used, but the <realm-id option> phrase is not used:
- (1) If the Schema does not specify a method of realm selection, then the DBCS uses an implementor defined method for selecting a realm from the realm names specified in the <within option> phrase. If the object record cannot occur in the selected realm, then the FORTRAN DML program is in error.

- (2) If the Schema specifies a method of realm selection, then that method is used. If the selected realm name is not referenced by a Sub-schema REALM statement, then the FORTRAN DML program is in error. Furthermore, if the selected realm name does not appear in the <within option> phrase, then the FORTRAN DML program is in error.

d. If both the <realm-id option> phrase and the <within option> phrase are used, then

- (1) A realm is selected as in described by Rule 8c in the previous paragraphs.

- (2) The name of the selected realm must appear in the <within option> phrase, and the contents of the <db name> phrase (if non-blank) must represent the name of the selected realm; otherwise the FORTRAN DML program is in error.

8. If a <record control> phrase is used, the data base procedure named by the <db proc> phrase will be called by the DBCS when the record type named in the <record declaration> phrase is accessed by the indicated DML statement type(s). See Section 3.1 rules 2a, 2b, and 2c for a description of the action of keywords BEFORE, AFTER and ERROR.

9. A <record lock option> phrase is used to specify individual privacy locks in the Sub-schema which must be satisfied by corresponding values in PRIVACY statements.

10. If the keyword ALL is used in the <record lock list> phrase, the action is as though all of the FORTRAN DML statement types named in the <record lock type> phrase definition were specified.

11. If no <record lock list> phrase is used, the <record lock option> phrase acts as though a <record lock list> phrase of FIND was specified.


```

*****
*
*   FORTRAN:
*
*   RECORD R1
*   RECORD R1(WITHIN(REALM1,REALM2),REALM=PLACE)
*   RECORD R1(LOCK(CONNECT,DISCONNECT)='LOCK1',
*             LOCK(ERASE)='OK TO ERASE')
*
*   COBOL:
*
*   RECORD SECTION.
*   01 R1.
*
*   RECORD SECTION.
*   01 R1 WITHIN REALM1, REALM2
*       REALM-ID IS PLACE.
*
*   RECORD SECTION.
*   01 R1 PRIVACY LOCK FOR CONNECT, DISCONNECT IS "LOCK1"
*       PRIVACY LOCK FOR ERASE IS "OK TO ERASE".
*
*****

```


3.6 TYPE-STATEMENTS

FUNCTION

A type-statement specifies the variables that are available to a FORTRAN Sub-schema. By virtue of its positional relationship with other type-statements within a data definition group, it specifies those variable names which constitute a named collection of related values. It can also specify privacy lock values to be associated with the use of the variable.

FORMAT

<type specification>[*<size constant>[,]]<type declaration>
[,<type declaration>] ...

<type specification> ::=

INTEGER|REAL|DOUBLE PRECISION|COMPLEX|LOGICAL|CHARACTER

<size constant> ::=

is an integer constant per FORTRAN [1], Section 4.3.1, whose value is greater than zero.

<type declaration> ::=

{<identifier>|<array declarator>}[*<size constant>]
[,<variable option>] ...

<array declarator> ::=

an array declarator as defined in FORTRAN [1] Section 5.1.

<variable option> ::=

LOCK(<variable lock list>)=<lock spec>

<variable lock list> ::=

ALL | {<variable lock type>[,<variable lock type>]... }

<variable lock type> ::=

STORE|GET|MODIFY|FETCH

<lock spec> ::=

<char con>|<db proc>

RULES

1. The optional <size constant> phrase is permitted only with a CHARACTER <type specification> phrase. A <size constant> phrase immediately following the word CHARACTER is the length specification for each entity in the statement not having its own length specification. A

<size constant> phrase must immediately follow an entity if the length specification is for only that entity. Note that for an array, the length specified is for each array element. If a length is not specified for an entity its length is one character.

2. A declaration is established for each <identifier> phrase. This declared name may appear in a syntactic context which requires a variable name. A declared variable name is given a data type as specified by the <type specification> which immediately precedes that declared name.
3. At run-time, the execution of a STORE or MODIFY statement can cause a mapping to occur which converts the User Working Area value instances of an affected variable from the characteristics as described by its Sub-schema declaration to value instances in the data base which have the characteristics of the Schema data item. Conversely, the execution of a GET or FETCH statement causes a mapping to occur which converts the value instances of an affected Schema data item in the data base from the characteristics described by the Schema to value instances in the User Working Area which have the characteristics described by its Sub-schema declaration. These conversions are performed exactly as described by the CODASYL Data Description Language JOD Section 3.3.14. For the purpose of interpreting these standard conversions, a LOGICAL data type is considered to have a representation which is a bit string consisting of one storage unit (See FORTRAN [1] Section 2.14), and the scale and precision of arithmetic data is defined by the implementor.
4. A <variable lock list> phrase is used to specify individual privacy locks in the Sub-schema which must be satisfied by corresponding values in PRIVACY statements.
5. If the keyword ALL is used in a <variable lock list> phrase, the action is as though a <variable lock list> phrase of FETCH, GET, MODIFY or STORE was supplied.
6. If no <variable lock list> phrase is used, the action is as though a <variable lock list> phrase of FETCH or GET was supplied.

```
*****
*
*   INTEGER A,B,C(2,3),D
*   REAL E,F(30),G(4,30),H,I
*   CHARACTER A*5,B,C(10,6),D(4)*5,E*2
*   CHARACTER*5,A,B*1,C(10,6)*1,D(4),E*2
*
*****
```


3.7 END STATEMENT

The END statement of the Sub-schema has the same attributes as the END statement in FORTRAN [1] Section 11.10.

4. DATA MANIPULATION LANGUAGE (DML)

The following paragraphs discuss the syntax, statement semantics, and pragmatics for each of the statements defined for FORTRAN DML usage.

See Section 5.3 for a discussion of the Data Base Exception Conditions which may be caused by execution of DML statements.

The following are the DML statements defined in this document:

ACCEPT
CONNECT
DISCONNECT
ERASE
FETCH
FIND
FINISH
GET
INVOKE
MODIFY
ORDER
PRIVACY
READY
STORE
USE

Withdrawn/Retirée

4.1 ACCEPT STATEMENTFUNCTION

The ACCEPT statement has two formats: one to access currency indicators, and one to provide the realm name of the record corresponding to a data base key value.

4.1.1 Format 1 - Accept Currency

```
ACCEPT( CURRENCY=<db key> [, <currency type>][, <error>] )
```

<currency type> ::=

```
{{RECORD|SET|REALM}=<db name>}|RUNUNIT|{KEY=<db key>}}
```

RULES FOR FORMAT 1

1. If the RECORD=<db name> clause is used, the data base key value corresponding to the currency indicator for that record type is moved into the variable indicated by the <db key> phrase.
2. If the SET=<db name> clause is used, the data base key value corresponding to the currency indicator for the current record of that set is moved into the variable indicated by the <db key> phrase.
3. If the REALM=<db name> clause is used, the data base key value corresponding to the currency indicator for the current record of that realm is moved into the variable indicated by the <db key> phrase.
4. If RUNUNIT is used, the data base key value corresponding to the currency indicator for the current record of the run unit is moved to the variable indicated by the <db key> phrase.
5. If none of the optional clauses are used, the data base key value corresponding to the currency indicator for the current record of the run unit is moved to the variable indicated by the <db key> phrase.

4.1.2 Format 2 - Accept Realm Name

```
ACCEPT( REALM NAME=<char var> [, <name type>][, <error>])
```

```
<name type> ::=
```

```
{ {RECORD|SET}=<db name> } | RUNUNIT | {KEY=<db key> }
```

RULES FOR FORMAT 2

1. If the RECORD=<db name> clause is used, the realm name of whatever record occurrence of the indicated record is current will be moved into the designated character variable.
2. If the SET=<db name> clause is used, the realm name of whatever occurrence of the indicated set is current will be moved into the designated character variable.
3. If RUNUNIT is used, the realm name for the current realm of the run unit will be moved into the indicated character variable.
4. If the KEY=<db key> clause is used, the realm name of the record associated with that data base key will be moved into the indicated character variable.
5. If none of the optional phrases are used, the current realm name of the run unit will be moved into the indicated character variable.

```
*****
*
* Format 1:  ACCEPT( CURRENCY=K1DB , RECORD=RECA)
*           ACCEPT( CURRENCY=K1DB , REALM=EMPFIL)
*           ACCEPT( CURRENCY=K1DB , SET=DEPT1)
*           ACCEPT( CURRENCY=K1DB , RUNUNIT)
* Format 2:  ACCEPT( REALM NAME=CHX , RECORD=RECA)
*           ACCEPT( REALM NAME=CHX , SET=DEPT1)
*           ACCEPT( REALM NAME=CHX , KEY=K1DB)
*           ACCEPT( REALM NAME=CHX , RUNUNIT)
*
*****
```


4.2 CONNECT STATEMENTFUNCTION

The CONNECT statement causes a record stored in the data base to become a member of one or more sets in which the record has been defined to be a MANUAL member.

FORMAT

```
CONNECT( [RECORD =<db name>,) {ALL|{SET=<db names>}} [,<error>] )
```

RULES

1. The realms in which all affected sets are stored, and in which the current record of the run unit is stored, must have been readied in update usage mode.
2. If the ALL option is used, the current record of the run unit must have been declared a MANUAL member in at least one set type defined in the Sub-schema, and each such set type becomes an object of the CONNECT.
3. If the ALL option is not used, the type of the current record of the run unit must have been declared a MANUAL member for each object set; i.e. any names in the SET=<db name> clause.
4. For each object set type, the object set occurrence is determined by the current record of the set, and conversely there must be a current record of the set for each object set type.
5. The current record of the run unit must not currently be a member of any object set, i.e. the current record of the run unit must not occur in the set in which a connect is being executed.
6. If all of the above conditions are satisfied, the current record of the run unit becomes a member of each set which is an object of the CONNECT.
7. The current record of the run unit is connected to each affected set in accordance with the set ordering criteria for each set type.
8. The current record of the run unit becomes the current record of each set type to which it is connected.
9. If the current record of the run unit cannot be connected to every set specified, it is not connected to any set and an exception condition is returned.


```
*****
*
*      CONNECT( RECORD=RECA, SET=(DEPT1,DEPT2))
*      CONNECT( RECORD=RECA, ALL)
*
*****
```

Withdrawn/Retirée

4.3 DISCONNECT STATEMENT

FUNCTION

The DISCONNECT statement logically removes a MANUAL OPTIONAL member record from one or more sets.

FORMAT

```
DISCONNECT( [RECORD=<db name>] [, {ALL|{SET=<db names>}}]
            [, <error>])
```

RULES

1. The realm in which the current record of the run unit is stored, and the realms in which affected sets are stored, must have been readied in update usage mode. All records containing derived data items that are affected by the execution of a DISCONNECT statement must also be stored in realms that have been readied in update usage mode.
2. If the SET=<db names> clause is used, the current record of the run unit must have been defined as a MANUAL OPTIONAL member of all specified sets, and the execution of a DISCONNECT statement causes it to be removed from membership of each of these sets.
3. If the ALL option is used, the current record of the run unit is removed from membership in all sets in which its record type is defined as a MANUAL OPTIONAL member.
4. Execution of a DISCONNECT statement does not alter currency indicators.
5. If the current record of the run unit cannot be disconnected from every set specified, it is not disconnected from any set and an exception condition is returned.

```
*****
*
*      DISCONNECT( RECORD=RECA, ALL)
*      DISCONNECT( RECORD=RECA, SET=(DEPT1,DEPT2))
*
*****
```


4.4 ERASE STATEMENTFUNCTION

The ERASE statement logically removes one or more records from the data base.

FORMAT

```
ERASE( [RECORD=<db name>],[<member type>][ ,MEMBERS=<db names>]
      [, SET=<db names>][, <error>] )
```

<member type> ::=

PERMANENT|SELECTIVE|ALL

RULES

1. The realms in which all affected records and sets are stored must have been in readied update usage mode. All records containing derived data items that are affected by the execution of an ERASE command must also be stored in realms that are in update usage mode.
2. The current record of the run unit is logically removed from the data base and is no longer accessible, and if PERMANENT, SELECTIVE or ALL is specified, more records may be erased.
3. If neither PERMANENT, SELECTIVE, nor ALL are used, the current record of the run unit must not be the owner of a set that currently has members; i.e. if it does have members an exception condition occurs.
4. If PERMANENT is specified, and the current record of the run unit is the owner of a set that currently has members:
 - a. Then each permanent member of that set is treated as though it were the current record of the run unit, and is logically removed from the data base. This process is repeated until all such records have been removed from the data base.
 - b. Any records in the affected sets which are transient members are not removed from the data base, but are treated as though they were the object of a DISCONNECT statement specifying that set name.
5. If SELECTIVE is used, the execution of an ERASE statement proceeds as though PERMANENT had been specified except:

If a MANUAL OPTIONAL member of an affected set is not a member of any other set, that member is logically removed from the data base.

6. If ALL is specified, and the current record of the run unit is the owner of a set that currently has members:
 - a. Then all members of that set are treated as though they were the current record of the run unit, and are logically removed from the data base.
 - b. All records in the affected sets which are MANUAL OPTIONAL members are not removed from the data base, but are treated as though they were the object of a DISCONNECT statement specifying that set name.
7. At the completion of the execution of an ERASE statement, the value of the run unit currency indicator is null. Other currency indicators are not affected.
8. If the execution of an ERASE statement is not able to erase or disconnect all object records, no records are affected, and an exception condition is returned.

```

*****
*
* ERASE( RECORD=RECA, SELECTIVE)
* ERASE( RECORD=RECA, PERMANENT)
* ERASE( RECORD=RECA, ALL)
* ERASE(RECORD=RECA, SELECTIVE, MEMBERS=(DEPT1,DEPT2),SET=DEPT3)*
*
*****

```


4.5 FETCH STATEMENTFUNCTION

The FETCH statement is a combined FIND and GET which establishes currency for the object record and makes available the contents of that record.

FORMAT

```
FETCH( {<rse>|{RSE=<char exp>}}[,<retain>][,<end>][,<error>] )
      [<item list>]
```

RULES

1. See Section 4.6 for a discussion of the FIND statement.
2. See Section 4.8 for a discussion of the GET statement.
3. The <end> phrase is meaningful only for record selection expressions using the keywords NEXT, PRIOR, or DUPLICATE. It is ignored for all other keywords.
4. If the execution of a FETCH statement is unsuccessful all currency indicators are returned to their status prior to the execution of the FETCH statement.

```
*****
*
*      FETCH(OWNER,SET=DEPT1) A,B,C
*      FETCH(NEXT,SET=FUNNY,END=123) X,Y,Z
*
*****
```


4.6 FIND STATEMENT

FUNCTION

The FIND statement is used to establish a specific record occurrence in the data base as the current record of the run unit.

FORMAT

FIND({<rse>|{RSE=<char exp>}}[, <retain>][,<end>][, <error>])

RULES

1. Execution of a FIND statement causes the record identified by the <rse> phrase to become the current record of the run unit. The realm name and the record name of that record are placed in the special registers DBRELM (data base realm name) and DBREC (data base record name) respectively.
2. If the RETAINING clause is not used, the record identified by the <rse> phrase becomes the current record of its realm-name, the current record of its record-name, and the current member of all sets in which it is a tenant.
3. Execution of a FIND statement does not make the selected record available to the run unit. To make data items available to the run unit, a GET command must be executed.
4. If the RSE=<char exp> clause is used, dynamic determination of the <rse> phrase occurs during execution of the FIND command. The <char exp> phrase must contain the desired <rse> phrase at the time the FIND statement is executed.
5. The <end> phrase is meaningful only for record selection expressions using the keywords NEXT, PRIOR, or DUPLICATE. It is ignored for all other keywords.

```
*****
*
*   FIND(OWNER, SET=DEPT1, RETAINING=REALM)
*   FIND(OWNER, SET=DEPT1, RETAINING=MULTIPLE)
*   FIND(OWNER, SET=DEPT1, RETAINING=SET)
*   FIND(OWNER, SET=DEPT1, RETAINING=(SET=(DEPT1,DEPT2)))
*   FIND(OWNER, SET=DEPT1, RETAINING=RECORD)
*   FIND(OWNER, SET=DEPT1, RETAINING=(RECORD,REALM))
*   FIND(OWNER, SET=DEPT1, RETAINING=(RECORD,REALM,SET))
*   FIND(RSE=CHARX)
*   FIND(RSE='OWNER')
*   FIND(NEXT, SET=DEPT1, END=2, ERR=4)
*
*****
```


4.7 FINISH STATEMENTFUNCTION

The FINISH statement terminates the availability of one or more realms to the program.

FORMAT

FINISH({ALL|{SET=<db names>}|{REALM=<db names>}}[, <error>])

RULES

1. If ALL is specified, the ready mode of all realms is terminated.
2. If the REALM=<db names> clause is used, the ready mode of the indicated realms is terminated.
3. If the SET=<db names> clause is used, the ready mode of all realms which may contain tenants of sets named by the <db names> phrase is terminated.
4. If the execution of a FINISH statement is not able to terminate the ready mode of each affected realm, no realms are affected.
5. At the completion of a FINISH statement, all currency indicators which identify records stored in the affected realms are null.
6. The action taken if any realm is in ready mode at the time the run unit terminates is specified by the implementor.

```
*****
*                                     *
*          FINISH( ALL )              *
*          FINISH( REALM=(EMPFIL,PAYFIL)) *
*                                     *
*****
```


4.8 GET STATEMENTFUNCTION

The GET statement makes part or all of a data base record available to the program, i.e. values are moved from the data base to the variables of the FORTRAN program.

FORMAT

GET ([RECORD=<db name>][,<error>]) [<item list>]

RULES

1. Variables names in the <item list> phrase must be found in the current record of the run unit. Only those items named in the list become available to the program.
2. If the <item list> phrase is not used, all items in the current record of the run unit become available to the program.
3. If the RECORD=<db name> clause is used, the indicated name is compared to the name of the current record of the run unit. If the names are not the same, an exception condition results.
4. To GET a derived data item successfully, all records that contain data items required by the DBCS to generate the derived value must be stored in realms that have been readied.

```
*****
*
*          GET( RECORD=RECA) A,B,C
*
*****
```


4.9 INVOKE STATEMENTFUNCTION

The INVOKE statement declares to the FORTRAN system the name and privacy keys of the Sub-schema to be accessed.

FORMAT

```
INVOKE(SUBSCHEMA={<char con>|<subschemaname>},
      SCHEMA={<char con>|<schemaname>} {,PRIVACY=<char con>} )
```

RULES

1. The PRIVACY phrase must be used if the Sub-schema is defined to have an invocation privacy lock.
2. The Sub-schema of the specified Schema becomes available subject to the following rule:

The PRIVACY expression is matched to the value of the invocation privacy lock associated with the indicated Sub-schema, according to the rules for comparison of alphanumeric operands in a relational condition. The Sub-schema is available to the program if, and only if, the values are equal.

3. A run unit may INVOKE only one Sub-schema at a time, but more than one program unit may INVOKE the same Sub-schema.
4. See Section 4.12 the PRIVACY statement for additional information.

```
*****
*
*      INVOKE( SUBSCHEMA='FDBML', SCHEMA='COMPX', PRIVACY='ABC')*
*      INVOKE(SUBSCHEMA=FDBML,SCHEMA=COMPX,PRIVACY='ABC')      *
*
*****
```


4.10 MODIFY STATEMENTFUNCTION

The MODIFY statement is used to change the contents of one or more data items in a record, and can also cause set occurrence relationship changes.

FORMAT

```
MODIFY([RECORD=<db name>][,ALL|{SET=<db names>}]  
      [,ONLY][,<error>])[<id list>]
```

RULES

1. This statement is used to alter the contents of one or more data items in the current record of the run unit.
2. If the RECORD=<db name> clause is used, the indicated record name must be the same name as the current record of the run unit.
3. If ONLY is specified the <id list> phrase is not applicable and no data items in the current run unit are changed in the data base.
4. If the optional <id list> phrase is not used, the contents of all the non-derived data items of the current record of the run unit are replaced by the values in the user variables.
5. The names in the <id list> phrase must reference non-derived data items contained within the current record of the run unit.
6. The realm containing the object record must have been readied in the update usage mode. Any records containing derived data items that are affected by the execution of a MODIFY statement must also be in readied mode.
7. If ALL is used, the object record's occurrence membership in every set in which it is a member might be changed.
8. If the SET=<db names> clause is used, the record's occurrence membership in the indicated sets is changed.
9. The sets to which the object record is reconnected are governed by set selection criteria. The object record is connected to each set occurrence in accordance with the set ordering criteria of each set. All sets in which membership is affected by execution of the MODIFY statement must be stored in realms that have been readied in update usage mode.

10. After successful execution the current record of the run unit becomes the current record of the set types specified.

```
*****  
*  
*          MODIFY( RECORD=RECA ) A,B,C          *  
*          MODIFY( RECORD=EMPLYE, SET=PAYROL )   *  
*          MODIFY( RECORD=EMPLYE, ALL )          *  
*  
*****
```


4.11 ORDER STATEMENTFUNCTION

The ORDER statement is used to reorder the members of a set logically.

FORMAT

```
ORDER({SET=<db name>}[,LOCALLY]{,<sort spec>} ... [,<error>] )
```

```
<sort spec> ::=
```

```
{ASCENDING|DESCENDING}=( RECORD|KEY|<sort field> ... )
```

```
<sort field> ::=
```

```
{ITEM=<id list>} | {KEY=<db names>} | {RECORD=<db names>}
```

RULES

1. The first <sort spec> phrase in the ORDER statement determines which member record types are to be sorted; these become the object records. A member type can be specified in one of the following ways:
 - a. If RECORD is used, all member record types of the set named in the ORDER statement are to be sorted.
 - b. If the RECORD=<db names> clause is used, occurrences of that record type are to be sorted.
 - c. If KEY is used, all member types of the indicated set are selected.
 - d. If the KEY=<db names> clause is used, the occurrences of the records named are sorted by their data base key values.
 - e. If the ITEM=<id list> clause is used, records which contain the indicated identifiers will be sorted.
2. The record, data base key, or identifiers named must occur within the designated set. The record names specified in the KEY record clause, and any records implicitly identified in this item clause, must be member record types within the designated set.
3. Within a <sort spec> phrase, the object record type may be specified only once. See 1.e above.
4. Records identified in subsequent <sort spec> phrases must be identified in the initial phrase.

5. Any data item or record name may appear only once.
6. Data names used in a <sort spec> phrase must not reference variable length or dimensioned entities.
7. The object records of an ORDER statement are the member occurrences of each specified member type in the set occurrence in which the current record of the set name is a tenant.
8. Any member records of the designated set that are not object records are logically placed at the end of the set. The relative ordering of the non-object records is not affected.
9. Each occurrence of the <sort spec> phrase indicates a sort key clause. Sort key clauses are listed from left to right in order of decreasing significance; i.e. the leftmost key is the major key.
 - a. When ASCENDING is used, the ordered sequence will be from the lowest value of the keys to the highest value, according to the rules for comparison in a relation condition.
 - b. When DESCENDING is used, the ordered sequence will be from the highest value of the keys to the lowest value, according to the rules for comparison in a relation condition.
10. If the RECORD=<db names> clause is used, the values of the record names of the object records are the key values. The value of a record name is the unpadded character string that is the name; e.g. the value of RECA is 'RECA'.
11. If the KEY=<db names> clause is used, the values of the data base keys of the object records are the key values.
12. If the ITEM=<id list> clause is used, the values of the data items are the key values.
13. The values of all keys for a given record occurrence are treated as though they were juxtaposed and left-justified. If the size of the entire key for any object record is less than the size of the key for any other object record, the smaller is increased in size by the juxtaposition of low-values in the rightmost positions of the shorter.
14. LOCALLY must be specified if the Set Description Entry of the Sub-schema specifies that the set order is permanent.
15. If the set order is permanent (or LOCALLY is used), the logical reordering of the object records lasts only until the run unit terminates, or until the ready mode for all affected realms is terminated. At that time, the logical order of the object records reverts to the order prior to the execution of the ORDER statement.

16. If LOCALLY is not used, the logical reordering of the object records remains in effect in the data base after the run unit terminates.
17. If LOCALLY is used and the object records are in non-temporary realms, those realms must be in PROTECTED RETRIEVAL or PROTECTED UPDATE usage mode.
18. If LOCALLY is not used, all object records must be in realms that are in EXCLUSIVE UPDATE usage mode.
19. Object records that contain derived data items must be stored in realms that are in EXCLUSIVE UPDATE usage mode. See Section 8.19 for a discussion of derived data items.

```

*****
*
*      ORDER( SET=DEPT1, LOCALLY
*              ,ASCENDING=(RECORD=(NONEXM, EXEMPT, TEMPRY))
*              ,DESCENDING=(KEY=(RATE, EMPNUMB, GRADE)))
*
*****

```


4.12 PRIVACY STATEMENTFUNCTION

The PRIVACY statement is used to provide the keys for the lock(s) specified in the Schema or Sub-schema. Each format of the PRIVACY statement provides access to a specific DML command and/or data base structure. Note that this statement must be executed prior to execution of a READY statement for realms, sets, records, or items which have PRIVACY protection. PRIVACY statements executed after a READY do not take effect until the affected realm has been terminated with a FINISH statement and a subsequent READY statement is executed.

The <priv key> phrase specification may be either a <char exp> phrase or the name of a data base procedure which validates the user privacy authorization. The <priv key> phrase is used to condition the special register DBPRIV, which in turn is used by the DBCS to control access to the data base. The internal value or the data base procedure validates the user request, then the user may perform the desired DML statement or operation.

GENERAL FORMAT FOR APPENDING PRIVACY KEY OR PROCEDURE

<priv key> ::=

<priv enable> | <priv disable>

<priv enable> ::=

PRIVACY={ <char exp> | <subroutine name> }

<priv disable> ::=

PRIVACY

The <priv enable> phrase of a privacy key is used to establish a value in the special register DBPRIV, thereby allowing the user to access data protected by a PRIVACY statement in the Sub-schema. The <priv disable> phrase is used to revoke the access rights previously established by a <priv enable> phrase.

The PRIVACY statement is used as follows:

Format 1 - Privacy Display Access for Sub-schema

Format 2 - Privacy Access for Realm

Format 3 - Privacy Access for Record(s)

Format 4 - Privacy Access for Set(s)

Format 5 - Privacy Access for Item(s)

4.12.1 Format 1 - Privacy Display Access for Sub-schema

The SUBSCHEMA clause is used to obtain access to the data base Sub-schema.

```
PRIVACY( [SUBSCHEMA=<char con>], DISPLAY, <priv key> )
```

4.12.2 Format 2 - Privacy Access for Realm

This format is used to obtain permission to access the data base at the REALM level. The specified <priv key> phrase value is evaluated by the DBCS at READY time.

```
PRIVACY( [SUBSCHEMA={<char con>|<subschema name>}],  
        REALM=<db names> [,<usage>],<priv key> )
```

4.12.3 Format 3 - Privacy Access for Record(s)

This format is used to obtain permission to use one or more specific DML statements upon a specified data base record.

```
PRIVACY( [SUBSCHEMA={<char con>|<subschema name>}],  
        RECORD[=<db names>][,<rec dml stmt>] ...,<priv key>)
```

<rec dml stmt> ::=

CONNECT|DISCONNECT|STORE|<erase>|MODIFY|FIND|GET|FETCH

<erase> ::=

ERASE {PERMANENT|SELECTIVE|ALL}

RULES FOR FORMAT 3

1. The record types indicated in the <db names> phrase must have been declared in the Sub-schema.
2. If the <rec dml stmt> phrase is not used the statement refers to all <rec dml stmt> phrases.

4.12.3.1 Format 4 - Privacy Access for Set(s)

This format is used to obtain access to DML functions which operate on set structures.

```
PRIVACY( [SUBSCHEMA={<char con>|<subschema name>} ,]
        SET[=<db names>][,<set dml stmt> ] ...,<priv key> )
```

```
<set dml stmt> ::=
```

```
ORDER | CONNECT | DISCONNECT | <erase> | FIND | FETCH
```

```
<erase> ::=
```

```
ERASE { PERMANENT | SELECTIVE | ALL }
```

RULES FOR FORMAT 4

1. The set indicated in the <db names> phrase must have been declared in the Sub-schema.
2. If the <set dml stmt> phrase is not used, the statement refers to all the <set dml stmt> phrases.

4.12.4 Format 5 - Privacy Access for Item(s)

This format is used to satisfy the protection locks for item-level access. Protection at the item-level is defined to apply to individual FORTRAN variables.

```
PRIVACY([SUBSCHEMA={<char con>|<subschema name>} ,]
        [REALM=<db names> ,] [SET=<db names> ,] [<item dml stmt> ,] ...
        <priv key> ) <id list>
```

```
<item dml stmt> ::=
```

```
MODIFY | GET | FETCH
```


RULES FOR FORMAT 5

1. The <id list> phrase must reference program variables found within data base records described in the Sub-schema.
2. If none of the optional <item dml stmt> phrases are used within the PRIVACY statement, then the <priv key> phrase applies to all of the statement types.

```

*****
*
* PRIVACY STATEMENT EXAMPLES
*
* Format 1.  PRIVACY(SUBSCHEMA='ACC',DISPLAY,PRIVACY='AJE20')
*           PRIVACY(DISPLAY,PRIVACY='AJE21')
*
* Format 2.  PRIVACY(SUBSCHEMA='CA',REALM=WORLD,EXCLUSIVE,
*                 UPDATE,PRIVACY='LV')
*
*           PRIVACY(REALM='FILE6',CONCURRENT,RETRIEVAL,
*                 PRIVACY='JQ17')
*           PRIVACY(REALM,PROTECTED,RETRIEVAL,PRIVACY='BKEY')
*           PRIVACY(REALM,EXCLUSIVE,PRIVACY='XZ13')
*           PRIVACY(REALM,RETRIEVAL,PRIVACY='ABC')
*           PRIVACY(REALM,PRIVACY='XRAY')
*
* Format 3.  PRIVACY(SUBSCHEMA=MAS,RECORD=('TOTALS',TAXES),
*                 CONNECT,STORE,PRIVACY='LOCKM')
*           PRIVACY(RECORD=TAXES,CONNECT,PRIVACY='LOCKM')
*           PRIVACY(RECORD,CONNECT,DISCONNECT,STORE,
*                 PRIVACY='LOCKM')
*           PRIVACY(RECORD,PRIVACY='LOCKM')
*
* Format 4.  PRIVACY(SUBSCHEMA=SLOTS,SET=JACKPOT,CONNECT,
*                 PRIVACY='LOCKM')
*           PRIVACY(SET=(ID,STREET,CITY),ORDER,PRIVACY='XYZ')
*           PRIVACY(SET,ORDER,CONNECT,DISCONNECT,PRIVACY='K')
*           PRIVACY(SET,PRIVACY='007')
*
* Format 5.  PRIVACY(SUBSCHEMA='PERSONNEL',MODIFY,FETCH,
*                 PRIVACY='AC146')SALARY,EXEMPT
*           PRIVACY(GET,PRIVACY='FH6036')ADDRESS,CITY
*           PRIVACY(PRIVACY='NEVADA')CITIES
*
*****

```


4.13 READY STATEMENTFUNCTION

The READY statement prepares one or more realms for processing.

FORMAT

READY({ALL|{SET=<db names>}}|{REALM=<db names>}},<usage>[,<error>])

RULES

1. Any realm defined as a temporary realm, or any set in a temporary realm, must be readied in EXCLUSIVE UPDATE usage mode.
2. No realm affected by the READY statement may already be in a readied state at the beginning of the READY statement.
3. Execution of a READY statement affects entire realms, regardless of the format of that statement. If ALL is used, all realms described by the Sub-schema are placed in ready mode.
4. If ALL is used, all realms described by the Sub-schema are placed in ready mode.
5. If the SET=<db names> clause is used, all realms which contain tenants of the named sets are placed in ready mode.
6. If the REALM=<db names> clause is used, each indicated realm is placed in ready mode.
7. The <usage> phrase establishes the usage mode for affected realms, thereby specifying the type of operations permitted. A usage mode remains in effect for affected realms until a FINISH statement for that realm is executed. See the FINISH statement for actions when a run unit is terminated.
8. If the <usage> phrase is not used, the execution of a READY statement proceeds as though CONCURRENT RETRIEVAL were specified.
9. Execution of a READY statement must follow the execution of all PRIVACY statements that are associated with all set types, all record types, all data items, and all DML operations that are permitted by the specified usage mode. In addition, the PRIVACY statements with the affected realms themselves must have been executed.

10. If any privacy lock is modified, that modification has no effect until the execution of the next READY statement. At that time, the new privacy lock is placed into effect for this run unit immediately prior to the execution of any privacy key validation caused by the READY statement.
11. The <usage> phrase specified for a realm must not conflict with the usage mode of that realm as already established by a concurrent run unit.
12. If the execution of a READY statement is unable to place all referenced realms in a readied mode, no realms are affected.

```

*****
*
*      READY( REALM=EMPL, PROTECTED, UPDATE )      *
*      READY( SET=DEPT1 , EXCLUSIVE, UPDATE )      *
*      READY( SET=DEPT1 , PROTECTED, UPDATE )      *
*      READY( SET=DEPT1 , EXCLUSIVE, RETRIEVAL )    *
*      READY( SET=DEPT1 , CONCURRENT, RETRIEVAL )   *
*
*****

```


4.14 STORE STATEMENTFUNCTION

The STORE statement causes a record to be stored in the data base.

FORMAT

STORE(RECORD=<db name>[,<retain>][,<error>])

RULES

1. The realm in which the indicated record is to be stored must have been readied in update usage mode. Any sets in which the record becomes a member as a result of execution of the STORE statement must be stored in realms that are in update usage mode. All records containing derived data items that are affected by the STORE statement must also be stored in realms that are in update usage mode.
2. The tenant records of all sets referenced by the set selection criteria must be in realms that are in ready mode.
3. The record being stored becomes a member of each set in which it has been declared an automatic member.
4. If the record being stored can be stored in more than one realm (i.e. it is declared in the Schema with a WITHIN clause of the form "WITHIN REALM1, REALM2, ... REALMID IS ID1"), the value of its REALMID when the STORE statement is executed must be one of the realms specified in its WITHIN clause. If the REALMID has a valid value, the record will be stored in the realm so specified.
5. The record being stored becomes the owner of an empty set for each set type in which it has been declared an owner.
6. The record being stored becomes a member of each set in accordance with its set selection criteria and its set ordering criteria.
7. A STORE statement causes the indicated record being stored to become the current record of the run unit. The realm-name and record-name for that record are placed in special registers DBRELM and DBREC, respectively.
8. If the <retain> phrase is not used, the indicated record becomes the current record of its realm, the current record of its record type, and the current record of all set types in which it is declared to be a tenant and current to run unit.

9. The data items of the user working area for the record being stored remain unchanged.

10. The record being stored is given a unique data base key value. The user, by means of the record's location mode, can specify certain constraints to be placed on the DBCS in its assignment of a data base key value.

a. If the record has no location mode, the DBCS uses an implementor-defined method of assigning a value.

b. If the record has a set location mode, the DBCS assigns a value as though the record was to become a member of a set of the set type associated with the record's location mode. Both the set selection criteria and the set ordering criteria are used by the DBCS in assigning a data base key value, in order to facilitate clustering.

c. If the record has a direct location mode, the contents of the location mode data item associated with the record are used to assign a data base key value, subject to the constraints listed below.

(1) If the location mode data item is null, the DBCS uses an implementor-defined method of assigning a value.

(2) If the location mode data item contains a value that does not specify an existing record in the data base, the DBCS assigns that value.

(3) If the location mode data item contains a value that does specify an existing record in the data base, the DBCS assigns the next logical value that does not specify an existing record in the data base.

d. If the record has a calculation location mode, the contents of the location mode data item associated with the record are used to assign a data base key value.

```
*****
*
*   STORE( RECORD=RECA )
*   STORE( RECORD=RECA, RETAINING=SET)
*   STORE( RECORD=RECA, RETAINING=(RECORD,SET=(DEPT1))
*   STORE( RECORD=RECA, RETAINING=MULTIPLE)
*
*****
```


4.15 USE STATEMENTFUNCTION

The USE statement establishes the subroutine procedure which is to be called in the event that a data base exception condition is encountered. The properties of USE-procedures are discussed in Section 5.2.1.

FORMAT

```
USE(PROCEDURE=<identifier> [, {ALL|OTHER|{STATUS=<status list>}}])
```

RULES

1. Execution of a USE statement causes the procedure specified by the <identifier> phrase to become the USE-procedure for each exception status code.
2. The procedure to be called is the subroutine named by the <identifier> phrase.
3. If ALL is used, the indicated procedure will be called for all data base exception processing.
4. If the STATUS=<status list> clause is used, the indicated procedure will be called whenever any of the conditions in the <status list> phrase are encountered.
5. If OTHER is used, the indicated procedure will be called for any data base exception conditions which are not previously established.

```
*****
*
*      USE(PROCEDURE=NOROOM, STATUS=('80100','90200','80300'))
*      USE(PROCEDURE=LOCKED, STATUS='90100')
*      USE(PROCEDURE=DBXCPT, OTHER )
*      USE(PROCEDURE=DBALL, ALL)
*
*****
```


5. EXCEPTION HANDLING

A Data Base Exception occurs when the Data Base Control System (DBCS) is unable to execute a FORTRAN DML statement successfully. The presence (or absence) of a Data Base Exception controls the flow of several Data Base Facility mechanisms.

1. A FORTRAN program must be notified that an exception has (or has not) occurred, and this is done by the DBCS placing appropriate values in one or more of the Data Base Special Registers. The Special Registers are described in Section 5.1.
2. If there is an associated USE condition/procedure for the exception which has occurred, the DBCS must cause it to be executed.
3. Similarly, if the unsuccessful DML statement contained an <error> phrase, the DBCS must cause control to pass to the appropriate statement or subroutine. The USE and <error> phrase mechanisms are described in Section 5.2.

5.1 DATA BASE SPECIAL REGISTERS

Data Base Special Registers are compiler-generated storage areas whose purpose is to store information for use by the program and DBCS when using the Data Base Facility.

The names of the Special Registers are neither keywords nor reserved words. They are merely normal variable names that "happen" to be supplied by the compiler (rather than the programmer) when the program unit being compiled contains any DML statements. Thus, such a program unit cannot explicitly declare any of these names.

5.1.1 DBSTAT - Data Base Status Register

Only one DBSTAT is allocated per run unit. The data type is CHARACTER*7. The value of DBSTAT may be accessed by any executable FORTRAN statement, but only the DBCS may change its value. The form of DBSTAT is that the first two bytes contain the statement code (table 5-1) and the next five bytes contain the status code (table 5-2).

The execution of a DML statement causes the DBCS to place a value into DBSTAT. This value, known as a Data Base Status Indicator, is conceptually made up of a Statement Code in the two high-order digits and a Status Code in the five low-order digits. If the execution of any DML statement results in a Data Base Exception Condition, the Statement Code indicates which type of DML statement caused the exception, and the Status Code indicates which type of Data Base Exception Condition has occurred. If the execution of a DML statement does not result in a Data Base Exception Condition, both the Statement Code and the Status Code of the Data Base Status Indicator are equal to zero, and execution of the statement is successfully completed.

5.1.2 DBRELM - Data Base Realm Register

Only one DBRELM is allocated per run unit. The data type is CHARACTER*30. The value of DBRELM may be accessed by any executable FORTRAN statement, but only the DBCS may change its value.

During the execution of a DML statement that is not successfully completed, the DBCS places the realm name of a realm associated with the Data Base Exception Condition in the special register DBRELM.

5.1.3 DBSET - Data Base Set Register

Only one DBSET is allocated per run unit. The data type is CHARACTER*30. The value of DBSET may be accessed by any executable FORTRAN statement, but only the DBCS may change its value.

During the execution of a DML statement that is not successfully completed, the DBCS places the set name of a set type associated with the Data Base Exception Condition in the special register DBSET.

5.1.4 DBREC - Data Base Record Register

Only one DBREC is allocated per run unit. The data type is CHARACTER*30. The value of DBREC may be accessed by any executable FORTRAN statement, but only the DBCS may change its value.

During the execution of a DML statement that is not successfully completed, the DBCS places the record name of a record type associated with the Data Base Exception Condition in the special register DBREC.

5.1.5 DBPRIV - Data Base Privacy Register

Only one DBPRIV is allocated per run unit. The data type is CHARACTER*10. The value in DBPRIV must be set by the FORTRAN object program to establish run unit authority to execute statements that may reference the data base. The value of DBPRIV may be changed only by a PRIVACY statement, but may be referenced in any portion of a program.

5.2 PROCEDURAL FLOW AFTER DATA BASE EXCEPTION

The actions of the DBCS upon detection of a data base exception condition are as follows:

1. Appropriate values are placed in the Data Base Special Registers.
2. If a data base procedure (See Section 2.3.8) specified in the Sub-schema applies to the attempted FORTRAN DML statement, it is executed.

3. The data base is placed in an appropriate state as defined in Section 4 for the various FORTRAN DML statements.
4. If an <error> phrase is present in the attempted FORTRAN statement, the <error> phrase is executed.
5. If a USE condition is specified in the FORTRAN program, and an <error> phrase is not used in the attempted DML statement, then the USE condition is executed.

5.2.1 Processing A USE Condition

When the DBCS has determined that there is a Data Base Exception and no <error> phrase, it examines the USE condition table maintained by the DBCS to see if the program has specified (see Section 4.15) a USE condition for the Data Base Exception which has occurred. If no USE condition is present, control returns to the statement following the unsuccessful DML statement. If a USE condition is present, control passes to the USE procedure. When the USE procedure is completed, control returns to the statement following the unsuccessfully executed DML statement. The implementor specifies whether control passes directly from the USE procedure to the statement or whether the USE procedure returns to the DBCS which itself passes control to this statement.

A USE procedure may contain any legal FORTRAN statement. However, what happens when a DML statement within a USE procedure cannot be successfully executed depends upon the implementor. For example, the procedure could activate a second USE procedure, abort the run-unit, or return immediately to the statement following the original unsuccessful DML statement.

5.2.2 Processing an <error> or <end> Phrase

If an <error> or <end> phrase is present, control will be passed to it.

If the <error> or <end> phrase specifies a statement number, the DBCS in effect returns to the application program and then executes a GOTO to that statement.

If the <error> or <end> phrase specifies a procedure name, processing is as for a USE procedure: the procedure is executed, and control returns to the statement following the unsuccessfully executed DML statement when the procedure is finished.

5.3 DATA BASE STATUS INDICATORS

The set of Data Base Status Indicators can be described in terms of three tables: the defined Statement Codes, the defined Data Base Exception Conditions, and their allowed combinations.

Table 5-1 Statement Codes

* Statement	* Statement Code	*
* ACCEPT	01	*
* CONNECT	02	*
* DISCONNECT	03	*
* ERASE	04	*
* FETCH	16	*
* FIND	05	*
* FINISH	06	*
* GET	08	*
* MODIFY	11	*
* ORDER	12	*
* PRIVACY	18	*
* READY	13	*
* STORE	15	*
* USE	17	*

Table 5-2 Data Base Status Codes

* Status Code	* Data Base Exception Condition	*
* (Data Base Retrieval Exception Conditions)		
* 02100	* End of set or realm has been reached, and an attempt has been made to locate the next record.	*
* 02200	* An unavailable realm has been requested by the DBCS	*
* 02300	* No set can be located to satisfy the set selection criteria	*
* 02400	* No record can be located to satisfy the record selection expression	*
* (Currency Indicator Exception Conditions)		
* 03100	* Current record of realm, set type, or record type is null	*
* 03200	* Current record of the run unit is null	*
* 03300	* Current record of the run unit is not correct record type	*
* (Name Specification Exception Conditions)		
* 04100	* Data Base key value is inconsistent with realm-name	*
* 04200	* Record type or set type requested by the DBCS is not defined in the Sub-schema	*
* 04300	* Realm-name is invalid	*
* 04400	* Specified Data Base Status Indicator is not defined	*

Table 5-2 Data Base Status Codes (continued)

```

*****
* Status Code * (Data Item Value Exception Conditions) *
*****
* 05100 * Contents of data items are duplicated in *
* * data base *
* 05200 * Contents of data item are invalid *
* 05300 * Contents of data item are not within range *
* * specified in CHECK clause *
*****
* * (Deleted Record Exception Conditions) *
*****
* 07100 * Access to a deleted record is specified *
* 07200 * Deletion of a non-empty set is specified *
*****
* * (Membership Exception Conditions) *
*****
* 08100 * Object record is a member of the set *
* 08200 * Object record is a permanent member of the *
* * set *
* 08300 * Object record is not a member of the set *
* 08400 * Set members cannot be stored in both *
* * permanent and temporary realms *
* 08500 * Set membership change is required *
*****
* * (Ready Mode Exception Conditions) *
*****
* 09100 * Realm not in ready mode *
* 09200 * Invalid usage mode for realm *
* 09300 * Realm in ready mode *
* 09400 * Usage mode conflict in a concurrent run unit *
*****
* * Implementor defined *
*****
* 7xxxx * xxxxx may be any number *
*****
* * (Resource Allocation Exception Conditions) *
*****
* 80100 * Space in data base is exhausted *
* 80200 * Space in realm is exhausted *
* 80300 * DBCS memory space is exhausted *
* 80400 * No data base key values in the data base are *
* * available for assignment *
* 80500 * No data base key values in the realm are *
* * available for assignment *
*****
* * (Privacy Lock Exception Conditions) *
*****
* 90100 * Locked resource is specified *
* 90200 * Incorrect PRIVACY key specified *
* 90300 * Attempt made to disable an access type which *
* * has not been enabled. *
* 99000 * Locked resource is specified resulting in *
* * abnormal termination of the run unit *
*****

```


Table 5-3 Statement Code and Status Code Combinations

*	S	A	C	D	E	F	F	F	G	M	O	P	R	S	U
**	T	C	O	I	R	E	I	I	E	O	R	R	E	T	S
* *	M	C	N	S	A	T	N	N	T	D	D	I	A	O	E
*	N	E	N	C	S	C	D	I		I	E	V	D	R	
*S *	T	P	E	O	E	H		S		F	R	A	Y	E	
*T *			T	C	N			H		Y		C			
*A C				T	N							Y			
*T O *					E										
*U D *					C										
*S E **					T										

* 02100 *						X	X								
* 02200 *			X	X	X	X	X			X		X	X		
* 02300 *						X	X			X			X		
* 02400 *						X	X								
* 03100 *	X	X				X	X			X	X		X		
* 03200 *	X	X	X	X	X	X	X		X	X					
* 03300 *		X	X	X	X	X	X		X	X					
* 04100 *													X		
* 04200 *					X	X	X						X		
* 04300 *													X		
* 04400 *														X	
* 05100 *		X	X	X						X	X		X		
* 05200 *	X					X			X	X			X		
* 05300 *						X			X	X			X		
* 07100 *						X	X		X						
* 07200 *				X											
* 08100 *		X													
* 08200 *			X												
* 08300 *			X							X					
* 08400 *		X								X			X		
* 08500 *										X					
* 09100 *		X	X	X	X	X	X			X	X		X		
* 09200 *		X	X	X						X	X		X		
* 09300 *												X			
* 09400 *												X			
* 80100 *		X	X	X						X	X		X		
* 80200 *		X	X	X						X	X		X		
* 80300 *		X	X	X		X	X	X	X	X	X				
* 80400 *													X		
* 80500 *													X		
* 90100 *		X	X	X	X	X			X	X	X		X	X	
* 90200 *												X			
* 90300 *												X			
* 99000 *		X	X	X	X	X			X	X	X		X	X	

6. STATEMENT SYNTAX SUMMARY6.1 Arguments

<argument> ::=

as defined in FORTRAN [1] Section 15.6.2.3.

6.2 Character Constant

<char con> ::=

as defined in FORTRAN [1] Section 4.8.1 with a maximum length not to exceed 30 characters.

6.3 Character Expression

<char exp> ::=

as defined in FORTRAN [1] Section 6.2.

6.4 Character Variable

<char var> ::=

as defined in FORTRAN [1] Section 8.4.2.

6.5 Data Base Key

<db key> ::=

integer variable as defined in FORTRAN [1] 8.4.1 with a non-negative value.

6.6 Data Base Name

<db name> ::=

<record name>|<set name>|<realm name>|<char exp>

```

*****
*                                                                 *
*      RECORD=<db name> phrase may be constructed as:          *
*                                                                 *
*      RECORD=EMPNO      -      a record name in the Sub-schema *
*      RECORD=AA2134     -      a CHARACTER variable            *
*      RECORD='EMP-NUMB' -      literal name of a record        *
*                           in the Sub-schema                    *
*                                                                 *
*****

```

6.7 Data Base Names

<db names> ::=

<db name>|(<db name> [,<db name>]...)


```

*****
*
*      REALM=ENG
*      REALM=( ENG )
*      REALM=( ENG, MFG )
*      REALM=( ENG, MFG, SHIP )
*      REALM=( 'ENG' , 'MFG' , SHIP )
*      REALM=( VARXYZ , VARABC , SHIP )
*
*****

```

6.8 Data Base Procedure

<db proc> ::=

 <subroutine name>[(<argument>)]

6.9 END Phrase

<end> ::=

 END={ <statement number> | { <subroutine name> [(<argument>)] } }

```

*****
*
*      END=999
*      END=DBESUB
*      END=RECOVR ( 210, DBRELM, DBREC )
*
*****

```

6.10 Error Phrase

<error> ::=

 ERR={ <statement number> | { <subroutine name> [(<argument>)] } }

```

*****
*
*      ERR=999
*      ERR=DBESUB
*      ERR=RECOVR ( 210, DBRELM, DBREC )
*
*****

```

6.11 Identifier

<identifier> ::=

as defined in FORTRAN [1] Section 2.5. This in turn refers to FORTRAN [1] section 2.2, where a symbolic name is defined.

6.12 Identifier Item

<id item> ::=

 <identifier>

6.13 Identifier List

<id list> ::=

 <id spec> [, <id spec>] ...

6.14 Identifier Specification

<id spec> ::=

 Input list item, as defined in 12.8.2.1 of FORTRAN [1], or
 Implied DO list as defined in 12.8.2 of FORTRAN [1].

6.15 Integer Constant

<integer> ::=

 as defined in FORTRAN [1] Section 4.3.1.

6.16 Item List

<item list> ::=

 <item spec> [,<item spec>] ...

6.17 Item Specification

<item spec> ::=

 Input list item as defined in FORTRAN [1] 12.8.2.1.

6.18 Realm Name

<realm name> ::=

 the name of a realm in the Sub-schema. See the <db name>
 phrase in Section 2.3.6 and Section 3.3 on the REALM
 statement.

6.19 Record Name

<record name> ::=

 the name of a Sub-schema record type. See the <db name>
 phrase in Section 2.3.6 and Section 3.5 on the RECORD
 statement.

6.20 Retaining Clause

<retain> ::=

<retain1>|<retain2>

FORMAT 1. Retaining Individual Currency Indicators.

<retain1> ::=

RETAINING=([RECORD][,REALM][,SET|{SET=<db names>}})

FORMAT 2. Retaining all Currency indicators

<retain2> ::=

RETAINING= MULTIPLE

```

*****
*
*      RETAINING=MULTIPLE
*      RETAINING=RECORD
*      RETAINING=REALM
*      RETAINING=( RECORD )
*      RETAINING=( RECORD, REALM )
*      RETAINING=( SET=EXEMPT )
*      RETAINING=( SET=( HOURLY , EXEMPT ) )
*      RETAINING=( RECORD, SET=( HOURLY , EXEMPT ) )
*      RETAINING=( RECORD, REALM, SET=( HOURLY , EXEMPT ) )
*
*****

```

6.21 Schema Item Name

<schema item name> ::=

the name of a data item in the Schema.

6.22 Schema Name

<schema name> ::=

the name of an object Schema. This name follows the rule for an <identifier> phrase.

6.23 Set Name

<set name> ::=

the name of a Sub-schema set type. See the <db name> phrase in Section 2.3.6 and Section 3.4 on the SET statement.

6.24 Statement Number

<statement number> ::=

as defined in FORTRAN [1] Section 3.4.

6.25 Status List

<status list> ::=

a sequence of character constants enclosed in quotes and separated by commas. See Section 4.15.

6.26 Subroutine Name

<subroutine name> ::=

as defined in FORTRAN [1] Section 18.2.3.

6.27 Sub-schema Name

<subschema name> ::=

the name of an object Sub-schema. This name follows the rules for an <identifier> phrase.

6.28 Usage

<usage> ::=

[PROTECTED|EXCLUSIVE|CONCURRENT][, RETRIEVAL|UPDATE]

6.29 RECORD SELECTION EXPRESSIONS

<rse> ::=

<rse 1>|<rse 2>|<rse 3>|<rse 4>|<rse 5>|<rse 6>|<rse 7>

Format 1 - <rse 1> Direct Access

Format 2 - <rse 2> Calculate Mode Access

Format 3 - <rse 3> Set Search Access

Format 4 - <rse 4> Positional Access

Format 5 - <rse 5> Currency Indicator Access

Format 6 - <rse 6> Set Owner Access

Format 7 - <rse 7> Set Occurrence Selection Rules Access

6.30 Format 1 - Direct Access

<rse 1> ::=

KEY=<db key> [, RECORD=<db name>]

6.31 Format 2 - Calculate Mode Access

<rse 2> ::=

{DUPLICATE|ANY} , RECORD=<db name>

6.32 Format 3 - Set Search Access

<rse 3> ::=

DUPLICATE, SET=<db name> , USING=<id list>

6.33 Format 4 - Positional Access

<rse 4> ::=

<posit>[,RECORD=<db name>]{,{REALM|SET}=<db name>}

<posit> ::=

{OFFSET=<int exp>}|FIRST|LAST|NEXT|PRIOR

6.34 Format 5 - Currency Indicator Access

<rse 5> ::=

CURRENT[,RECORD=<db name>][, {REALM|SET}=<db name>]

6.35 Format 6 - Set Owner Access

<rse 6> ::=

OWNER, SET=<db name>

6.36 Format 7 - Set Occurrence Selection Rules Access

<rse 7> ::=

RECORD=<db name>,[CURRENT] SET=<db name> [,USING=<id list>]


```

*****
*
*   RECORD SELECTION EXPRESSIONS EXAMPLES
*
*   Format 1.      (KEY=DBK, RECORD=RECA)
*
*   Format 2.      (ANY,          RECORD=RECA)
*                  (DUPLICATE, RECORD=RECA)
*
*   Format 3.      (DUPLICATE, SET=DEPT1, USING=(A,B,C))
*
*   Format 4.      (OFFSET=1   , RECORD=RECA, SET=DEPT1)
*                  (OFFSET=-5 , RECORD=RECA, REALM=EMPFIL)
*                  ( FIRST    , RECORD=RECA, SET=DEPT1)
*                  ( LAST     , RECORD=RECA, SET=DEPT1)
*                  ( NEXT     , RECORD=RECA, SET=DEPT1)
*                  ( PRIOR    , RECORD=RECA, SET=DEPT1)
*
*   Format 5.      (CURRENT, RECORD=RECA, SET=DEPT1)
*                  (CURRENT,          REALM=EMPFIL)
*                  (CURRENT)
*
*   Format 6.      (OWNER, SET=DEPT1)
*
*   Format 7.      (RECORD=RECA, CURRENT SET=DEPT1, USING=(A,B))
*                  (RECORD=RECA,      SET=DEPT1, USING=ABCDE)
*
*****

```


6.37 TENANCY CONDITIONS

<tenancy condition> ::=

{MEMBER(<db name>)} | {OWNER(<db name>)} | {TENANT(<db name>)}

```

*****
*
*      IF(      MEMBER(EXEMPT)) GO TO 101
*      IF(.NOT. MEMBER(EXEMPT)) GO TO 102
*      IF(      OWNER  (EXEMPT)) GO TO 103
*      IF(.NOT. OWNER  (EXEMPT)) GO TO 104
*      IF(      TENANT(EXEMPT)) GO TO 105
*      IF(.NOT. TENANT(EXEMPT)) GO TO 106
*
*****

```

6.38 MEMBERSHIP CONDITIONS

<member condition> ::=

{EMPTY(<db name>)} | {FIRST(<db name>)} | {LAST(<db name>)}

```

*****
*
*      IF(      EMPTY(EXEMPT)) GO TO 101
*      IF(.NOT. EMPTY(EXEMPT)) GO TO 102
*      IF(      FIRST(EXEMPT)) GO TO 103
*      IF(.NOT. FIRST(EXEMPT)) GO TO 104
*      IF(      LAST  (EXEMPT)) GO TO 105
*      IF(.NOT. LAST  (EXEMPT)) GO TO 106
*
*****

```

6.39 NULLITY CONDITION

<null cond> ::=

NULL (<identifier>)

```

*****
*
*      IF(      NULL(TAXES)) GO TO 101
*      IF(.NOT. NULL(TAXES)) GO TO 102
*
*****

```


6.40 SUBSCHEMA STATEMENT

```
SUBSCHEMA {<char con>|<schema name>} ,SCHEMA={<char con>|<schema name>}
  [,<subschema control>] ... [,<subschema lock option>] ...
  [,<subschema key option>]
```

```
<subschema control> ::=
```

```
    CALL <db proc> {BEFORE|AFTER|ERROR} {INVOKE|FINISH}
```

```
<subschema lock option> ::=
```

```
    LOCK(<lock list>)=<lock spec>
```

```
<lock list> ::=
```

```
    ALL | {<subschema lock type>[,<subschema lock type>]... }
```

```
<subschema lock type> ::=
```

```
    INVOKING | ALTERING | LOCKS
```

```
<lock spec> ::=
```

```
    <char con>|<db proc>
```

```
<subschema key option> ::=
```

```
    KEY=<lock spec>
```

```
*****
*
*  FORTRAN:
*
*  SUBSCHEMA SUB1,SCHEMA='DATA-BASE'
*  SUBSCHEMA SUB1,SCHEMA='DATA-BASE',KEY='DOIT'
*  SUBSCHEMA SUB1,SCHEMA='DATA-BASE',
*      CALL UMONTR BEFORE INVOKE,
*      CALL UCLOSE AFTER FINISH,
*      LOCK(INVOKING,LOCKS)='LOCK2'
*      LOCK(ALTERING)='OK TO ALTER'
*
*  COBOL:
*
*  SS SUB1 WITHIN DATA-BASE.
*  SS SUB1 WITHIN DATA-BASE
*      PRIVACY KEY IS "DOIT".
*  SS SUB1 WITHIN DATA-BASE
*      PRIVACY LOCK FOR INVOKING, LOCKS IS "LOCK2"
*      PRIVACY LOCK FOR ALTERING IS "OK TO ALTER".
*
*****
```


6.41 ALIAS STATEMENT

ALIAS [(<schema name type>)] <re-name> [,<re-name>]....

<schema name type> ::=

REALM|SET|RECORD|ITEM

<re-name> ::=

{<char con>|<identifier>} = <name in schema>

<name in schema> ::=

<char con>

The value of the <name in schema> phrase has the following syntax:

<realm name>|<set name>|<schema item name>|<record name>|
<record name>.<schema item name>

```

*****
*
*   FORTRAN:
*
*   ALIAS FIELD1='FIELD-1'
*   ALIAS DATE2='REC1.DATE'
*   ALIAS(RECORD) PRIME='GROUP'
*   ALIAS AMT='PAY-AMOUNT',TAX='LOCAL-TAX'
*   ALIAS 'P-AMT' = 'PAY-AMOUNT'
*
*   COBOL:
*
*   MAPPING DIVISION.
*   ALIAS SECTION.
*       AD ==FIELD-1== BECOMES FIELD1.
*       AD ==REC1;DATE== BECOMES DATE2.
*       AD ==GROUP== RECORD-NAME BECOMES PRIME.
*       AD ==LOCAL-TAX==BECOMES TAX.
*
*****

```


6.42 REALM STATEMENT

REALM<realm declaration>[,<realm declaration>]...

<realm declaration> ::=

<db name> [<realm control>] ... [(<realm lock list>)]

<realm control> ::=

CALL <db proc> {BEFORE | AFTER | ERROR} <realm lock>

<realm lock> ::=

ALL | {[EXCLUSIVE|PROTECTED]{RETRIEVAL | UPDATE}}

<realm lock list> ::=

<realm lock option>[,<realm lock option>]...

<realm lock option> ::=

LOCK<realm lock type>=<lock spec>

<realm lock type> ::=

(<realm lock>[,<realm lock>]...)

<lock spec> ::=

<char con>|<db proc>

```
*****
*
*  FORTRAN:
*
*      REALM REALM1,REALM2
*      REALM REALM1
*          CALL PROC1 BEFORE EXCLUSIVE UPDATE (LOCK='LOCK1'),
*          REALM2(LOCK(EXCLUSIVE UPDATE)='EU',
*          LOCK(RETRIEVAL)='R')
*
*  COBOL:
*
*      REALM SECTION.
*          RD REALM1.
*          RD REALM2.
*
*      REALM SECTION.
*          RD REALM1 PRIVACY LOCK IS "LOCK1".
*          RD REALM2 PRIVACY LOCK FOR EXCLUSIVE UPDATE IS "EU"
*          PRIVACY LOCK FOR RETRIEVAL IS "R".
*
*****
```


6.43 SET STATEMENT

SET <set declaration> [,<set declaration>]...

<set declaration> ::=

<db name>[(<set option> [,<set option>]...)]

<set option> ::=

<set control>|<selection option>|<set lock option>

<set control> ::=

CALL <db proc> {BEFORE | AFTER | ERROR}
 { {INSERT|REMOVE}| {FIND|FETCH}
 [OWNER | NEXT | FIRST | LAST | PRIOR | VIASET | OFFSET]}

<selection option> ::=

SELECTION({RECORD=<db name> ,ENTRY <db name>=<entry specifier>
 [,<path specifier>] ...} | DBPROC=<db proc>)

<entry specifier> ::=

SYSTEM|CURRENT|<calc entry>|DIRECT(<db key>)

<calc entry> ::=

CALC<key list>

<key list> ::=

(<key item>[,<key item>] ...)

<key item> ::=

<id item>[=<identifier>]

<path specifier> ::=

PATH = (<db name>[,<key item>]...)

<set lock option> ::=

LOCK<set lock list>=<lock spec>

<set lock list> ::=

ALL | {(<set lock type>[,<set lock type>]...)}>

<set lock type> ::=

CONNECT|DISCONNECT|FIND|ORDER|MODIFY|FETCH

<lock spec> ::=

<char con>|<db proc>


```
*****
*
*  FORTRAN:
*
*  SET SET1, SET2, SET3
*  SET SET1(LOCK(CONNECT, FIND)='CF-LOCK')
*  SET SET1(SELECTION(RECORD=REC1,
*    PATH SET2(S2=S3, S4=S5), PATH SET3(S6=S7)))
*
*
*  COBOL:
*
*  SET SECTION.
*    SD SET1.
*    SD SET2.
*    SD SET3.
*
*  SET SECTION.
*    SD SET1 PRIVACY LOCK FOR CONNECT,
*    FIND IS "CF-LOCK".
*
*  SET SECTION.
*    SD SET3
*      SET SELECTION FOR REC1 IS
*      VIA SET1 OWNER DBK1
*      VIA SET2 OWNER VALUE OF S2 IS S3  S4 IS S5
*      VIA SET3 OWNER VALUE OF S6 IS S7.
*
*****
```


6.44 RECORD STATEMENT

RECORD <record declaration>[,<record declaration>] ...

<record declaration> ::=

<db name>[(<record option> [,<record option>] ...)]

<record option> ::=

<record control>|<within option>|<realm-id option>|
<record lock option>|<check option>

<record control> ::=

CALL <db proc> { BEFORE | AFTER | ERROR } <record lock list>

<record lock list> ::=

ALL | { <record lock type>[,<record lock type>]... }

<record lock type> ::=

CONNECT|DISCONNECT|STORE|DELETE|MODIFY|FIND|GET|FETCH|
{ ERASE [ALL|PERMANENT|SELECTIVE] }

<within option> ::=

WITHIN<db names>

<realm-id option> ::=

REALMID=<char var>

<record lock option> ::=

LOCK(<record lock list>)=<lock spec>

<lock spec> ::=

<char con>|<db proc>

<check option> ::=

CHECK={<db proc> | DATA TYPE}


```
*****
*
*  FORTRAN:
*
*  RECORD R1
*  RECORD R1(WITHIN(REALM1, REALM2), REALM=PLACE)
*  RECORD R1(LOCK(CONNECT, DISCONNECT)='LOCK1',
*            LOCK(ERASE)='OK TO ERASE')
*
*  COBOL:
*
*  RECORD SECTION.
*  01 R1.
*
*  RECORD SECTION.
*  01 R1 WITHIN REALM1, REALM2
*      REALM-ID IS PLACE.
*
*  RECORD SECTION.
*  01 R1 PRIVACY LOCK FOR CONNECT, DISCONNECT IS "LOCK1"
*      PRIVACY LOCK FOR ERASE IS "OK TO ERASE".
*
*****
```


6.45 TYPE-STATEMENTS

<type specification>[*<size constant>[,]]<type declaration>
[,<type declaration>] ...

<type specification> ::=

INTEGER|REAL|DOUBLE PRECISION|COMPLEX|LOGICAL|CHARACTER

<size constant> ::=

is an integer constant per FORTRAN [1], Section 4.3.1, whose value is greater than zero.

<type declaration> ::=

{<identifier>|<array declarator>}<size constant>
[,<variable option>] ...

<array declarator> ::=

an array declarator as defined in FORTRAN [1] Section 5.1.

<variable option> ::=

LOCK(<variable lock list>)=<lock spec>

<variable lock list> ::=

ALL | {<variable lock type>[,<variable lock type>]... }

<variable lock type> ::=

STORE|GET|MODIFY|FETCH

<lock spec> ::=

<char con>|<db proc>

```
*****
*
*   INTEGER A,B,C(2,3),D
*   REAL E,F(30),G(4,30),H,I
*   CHARACTER A*5,B,C(10,6),D(4)*5,E*2
*   CHARACTER*5,A,B*1,C(10,6)*1,D(4),E*2
*
*****
```


6.46 ACCEPT STATEMENT6.47 Format 1 - Accept Currency

```
ACCEPT( CURRENCY=<db key> [, <currency type>][, <error>] )
```

```
<currency type> ::=
```

```
{ {RECORD|SET|REALM}=<db name> } | RUNUNIT | {KEY=<db key> }
```

6.48 Format 2 - Accept Realm Name

```
ACCEPT( REALM NAME=<char var> [, <name type>][, <error>])
```

```
<name type> ::=
```

```
{ {RECORD|SET}=<db name> } | RUNUNIT | {KEY=<db key> }
```

```
*****
*
* Format 1:  ACCEPT( CURRENCY=K1DB , RECORD=RECA)
*           ACCEPT( CURRENCY=K1DB , REALM=EMPFIL)
*           ACCEPT( CURRENCY=K1DB , SET=DEPT1)
*           ACCEPT( CURRENCY=K1DB , RUNUNIT)
* Format 2:  ACCEPT( REALM NAME=CHX , RECORD=RECA)
*           ACCEPT( REALM NAME=CHX , SET=DEPT1)
*           ACCEPT( REALM NAME=CHX , KEY=K1DB)
*           ACCEPT( REALM NAME=CHX , RUNUNIT)
*
*****
```


6.49 CONNECT STATEMENT

```
CONNECT( [RECORD =<db name> ,] {ALL|{SET=<db names>}} [,<error>] )
```

```
*****
*
*          CONNECT( RECORD=RECA, SET=(DEPT1,DEPT2))
*          CONNECT( RECORD=RECA, ALL)
*
*****
```

6.50 DISCONNECT STATEMENT

```
DISCONNECT( [RECORD=<db name>] [, {ALL|{SET=<db names>}}]
            [,<error>] )
```

```
*****
*
*          DISCONNECT( RECORD=RECA, ALL)
*          DISCONNECT( RECORD=RECA, SET=(DEPT1,DEPT2))
*
*****
```

6.51 ERASE STATEMENT

```
ERASE( [RECORD=<db name> ,] [<member type>] [ ,MEMBERS=<db names>]
      [, SET=<db names>] [, <error>] )
```

<member type> ::=

PERMANENT | SELECTIVE | ALL

```
*****
*
* ERASE( RECORD=RECA, SELECTIVE)
* ERASE( RECORD=RECA, PERMANENT)
* ERASE( RECORD=RECA, ALL)
* ERASE( RECORD=RECA, SELECTIVE, MEMBERS=(DEPT1,DEPT2), SET=DEPT3)
*
*****
```


6.52 FETCH STATEMENT

```
FETCH( {<rse>|{RSE=<char exp>}}[,<retain>][,<end>][,<error>] )
      [<item list>]
```

```
*****
*
*      FETCH(OWNER,SET=DEPT1) A,B,C
*      FETCH(NEXT,SET=FUNNY,END=123) X,Y,Z
*
*****
```

6.53 FIND STATEMENT

```
FIND( {<rse>|{RSE=<char exp>}}[, <retain>][,<end>][, <error>])
```

```
*****
*
*      FIND(OWNER, SET=DEPT1, RETAINING=REALM)
*      FIND(OWNER, SET=DEPT1, RETAINING=MULTIPLE)
*      FIND(OWNER, SET=DEPT1, RETAINING=SET)
*      FIND(OWNER, SET=DEPT1, RETAINING=(SET=(DEPT1,DEPT2)))
*      FIND(OWNER, SET=DEPT1, RETAINING=RECORD)
*      FIND(OWNER, SET=DEPT1, RETAINING=(RECORD,REALM))
*      FIND(OWNER, SET=DEPT1, RETAINING=(RECORD,REALM,SET))
*      FIND(RSE=CHARX)
*      FIND(RSE='OWNER')
*      FIND(NEXT,SET=DEPT1,END=2,ERR=4)
*
*****
```

6.54 FINISH STATEMENT

```
FINISH( {ALL|{SET=<db names>}}|{REALM=<db names>}}[, <error>] )
```

```
*****
*
*      FINISH( ALL )
*      FINISH( REALM=(EMPFIL,PAYFIL))
*
*****
```


6.55 GET STATEMENT

```
GET ( [RECORD=<db name>] [,<error>] ) [ <item list>]
```

```
*****
*
*           GET( RECORD=RECA ) A,B,C
*
*****
```

6.56 INVOKE STATEMENT

```
INVOKE ( SUBSCHEMA = { <char con> | <subschema name> },
        SCHEMA = { <char con> | <schema name> } { , PRIVACY = <char con> } )
```

```
*****
*
*   INVOKE( SUBSCHEMA='FDBML', SCHEMA='COMPL', PRIVACY='ABC' ) *
*   INVOKE( SUBSCHEMA=FDBML, SCHEMA=COMPL, PRIVACY='ABC' )    *
*
*****
```

6.57 MODIFY STATEMENT

```
MODIFY ( [RECORD=<db name>] [, ALL | { SET = <db names> } ]
        [, ONLY] [, <error>] ) [ <id list> ]
```

```
*****
*
*           MODIFY( RECORD=RECA ) A,B,C
*           MODIFY( RECORD=EMPLYE, SET=PAYROL )
*           MODIFY( RECORD=EMPLYE, ALL )
*
*****
```

6.58 ORDER STATEMENT

```
ORDER ( { SET = <db name> } [, LOCALLY] { , <sort spec> } ... [, <error> ] )
```

```
<sort spec> ::=
```

```
{ ASCENDING | DESCENDING } = ( RECORD | KEY | <sort field> ... )
```

```
<sort field> ::=
```

```
{ ITEM = <id list> } | { KEY = <db names> } | { RECORD = <db names> }
```

```
*****
*
*   ORDER( SET=DEPT1, LOCALLY
*           ,ASCENDING=(RECORD=(NONE XM, EXEMPT, TEMPRY))
*           ,DESCENDING=(KEY=(RATE, EMPNUMB, GRADE)))
*
*****
```


6.59 PRIVACY STATEMENTGENERAL FORMAT FOR APPENDING PRIVACY KEY OR PROCEDURE

<priv key> ::=

<priv enable> | <priv disable>

<priv enable> ::=

PRIVACY={ <char exp> | <subroutine name> }

<priv disable> ::=

PRIVACY

The PRIVACY statement is used as follows:

Format 1 - Privacy Display Access for Sub-schema

Format 2 - Privacy Access for Realm

Format 3 - Privacy Access for Record(s)

Format 4 - Privacy Access for Set(s)

Format 5 - Privacy Access for Item(s)

6.60 Format 1 - Privacy Display Access for Sub-schema

PRIVACY([SUBSCHEMA=<char con>], DISPLAY, <priv key>)

6.61 Format 2 - Privacy Access for Realm

PRIVACY([SUBSCHEMA={<char con>|<subschema name>}],
REALM=<db names> [,<usage>],<priv key>)

6.62 Format 3 - Privacy Access for Record(s)

PRIVACY([SUBSCHEMA={<char con>|<subschema name>}],
RECORD[=<db names>][,<rec dml stmt>] ...,<priv key>)

<rec dml stmt> ::=

CONNECT|DISCONNECT|STORE|<erase>|MODIFY|FIND|GET|FETCH

<erase> ::=

ERASE {PERMANENT|SELECTIVE|ALL}

6.63 Format 4 - Privacy Access for Set(s)

```

PRIVACY( [SUBSCHEMA={<char con>|<subschema name>} ,] SET[=<db
names>]
        [,<set dml stmt> ] ... ,<priv key> )

```

```

<set dml stmt> ::=

```

```

ORDER | CONNECT | DISCONNECT | <erase> | FIND | FETCH

```

```

<erase> ::=

```

```

ERASE { PERMANENT | SELECTIVE | ALL }

```

6.64 Format 5 - Privacy Access for Item(s)

```

PRIVACY([SUBSCHEMA={<char con>|<subschema name>},]
        [REALM=<db names>],] [SET=<db names>],] [<item dml stmt>],] ...
        <priv key> ) <id list>

```

```

<item dml stmt> ::=

```

```

MODIFY | GET | FETCH

```



```

*****
*
*   PRIVACY STATEMENT EXAMPLES
*
*   Format 1.   PRIVACY(SUBSCHEMA='ACC',DISPLAY,PRIVACY='AJE20')
*               PRIVACY(DISPLAY,PRIVACY='AJE21')
*
*   Format 2.   PRIVACY(SUBSCHEMA='CA',REALM=WORLD,EXCLUSIVE,
*                   UPDATE,PRIVACY='LV')
*
*               PRIVACY(REALM='FILE6',CONCURRENT,RETRIEVAL,
*                   PRIVACY='JQ17')
*               PRIVACY(REALM,PROTECTED,RETRIEVAL,PRIVACY='BKEY')
*               PRIVACY(REALM,EXCLUSIVE,PRIVACY='XZ13')
*               PRIVACY(REALM,RETRIEVAL,PRIVACY='ABC')
*               PRIVACY(REALM,PRIVACY='XRAY')
*
*   Format 3.   PRIVACY(SUBSCHEMA=MAS,RECORD=('TOTALS',TAXES),
*                   CONNECT,STORE,PRIVACY='LOCKM')
*               PRIVACY(RECORD=TAXES,CONNECT,PRIVACY='LOCKM')
*               PRIVACY(RECORD,CONNECT,DISCONNECT,STORE,
*                   PRIVACY='LOCKM')
*               PRIVACY(RECORD,PRIVACY='LOCKM')
*
*   Format 4.   PRIVACY(SUBSCHEMA=SLOTS,SET=JACKPOT,CONNECT,
*                   PRIVACY='LOCKM')
*               PRIVACY(SET=(ID,STREET,CITY),ORDER,PRIVACY='XYZ')
*               PRIVACY(SET,ORDER,CONNECT,DISCONNECT,PRIVACY='K')
*               PRIVACY(SET,PRIVACY='007')
*
*   Format 5.   PRIVACY(SUBSCHEMA='PERSONNEL',MODIFY,FETCH,
*                   PRIVACY='AC146')SALARY,EXEMPT
*               PRIVACY(GET,PRIVACY='FH6036')ADDRESS,CITY
*               PRIVACY(PRIVACY='NEVADA')CITIES
*
*****

```

6.65 READY STATEMENT

READY({ALL|{SET=<db names>}|{REALM=<db names>}},<usage>[,<error>])

```

*****
*
*   READY( REALM=EMPL, PROTECTED, UPDATE )
*   READY( SET=DEPT1 , EXCLUSIVE, UPDATE )
*   READY( SET=DEPT1 , PROTECTED, UPDATE )
*   READY( SET=DEPT1 , EXCLUSIVE, RETRIEVAL )
*   READY( SET=DEPT1 , CONCURRENT, RETRIEVAL )
*
*****

```


6.66 STORE STATEMENT

```
STORE( RECORD=<db name>[ ,<retain>][ ,<error>])
```

```
*****
*
*      STORE( RECORD=RECA )
*      STORE( RECORD=RECA, RETAINING=SET)
*      STORE( RECORD=RECA, RETAINING=(RECORD, SET=(DEPT1))
*      STORE( RECORD=RECA, RETAINING=MULTIPLE)
*
*****
```

6.67 USE STATEMENT

```
USE(PROCEDURE=<identifier> [, {ALL|OTHER|{STATUS=<status list>}}])
```

```
*****
*
*      USE(PROCEDURE=NOROOM, STATUS=('80100','90200','80300'))
*      USE(PROCEDURE=LOCKED, STATUS='90100')
*      USE(PROCEDURE=DBXCPT, OTHER )
*      USE(PROCEDURE=DBALL, ALL)
*
*****
```


7. REFERENCES

1. American National Standards Committee X3J3.
Draft Proposed ANS FORTRAN, ACM Sigplan Notices,
vol. 11, no.3, 1976 March.
2. CODASYL Data Base Task Group.
Data Base Task Group Report.
New York: Association for Computing Machinery,
(Apr., 1971), 269 pp.
3. CODASYL Data Description Language Committee.
Data Description Language - Journal of Development.
Washington: National Bureau of Standards, (1973),
136 pp.
4. CODASYL Systems Committee.
Feature Analysis of Generalized Data Base Management
New York: Association for Computing Machinery, (1971),
320 pp.
5. CODASYL Systems Committee.
Communications of the ACM, vol. 14, (May, 1971),
pp. 308-218.
6. CODASYL DBLTG Committee.
COBOL Journal of Development, U. S. Standard Serial Number
0591-0218, June 1975.
7. Martin, James.
Computer Data-Base Organization, New York, Prentice Hall,
577 pp., 1975.
8. Munson, Barry R. and Chester M. Smith, Jr.
"The Study of Data Base Management Systems with Bibliography."
in NBS Special Publication 451, DATA BASE DIRECTIONS:
The Next Steps, (September 1976), pp. 111-149.

8. GLOSSARY

8.1 Abnormal Termination

the premature end of execution of a program due to the detection, by the operating system or DBCS, of a situation that prevents successful execution of that program.

8.2 Alias Description Entry

the usage of the ALIAS statement in the Sub-schema that defines an alias to be used in the Sub-schema and FORTRAN program for a name defined in the Schema.

8.3 Automatic Member

a record that automatically becomes a member of a given set when the record is initially stored in the data base.

8.4 Available Mode

the state of a realm when it is accessible to the DBCS and is not under the exclusive control of a different run unit.

8.5 Calculation Key

When the LOCATION MODE is CALC, these are the data items defined in a record whose contents are used by the DBCS in its algorithm to calculate a data base key value when that record is stored.

8.6 Calculation Location Mode

the LOCATION MODE statement in the Schema where the user specifies to the DBCS the data items whose contents are to be used in the location mode algorithm.

8.7 Concurrent Run Unit

a run unit other than this run unit that has been initiated, but has not terminated, during the time in which this run unit has been initiated but not terminated.

8.8 Currency Indicator

a conceptual entity that is used by the DBCS as a place marker in accessing the data base.

8.9 Data Base Control System (DBCS)

an input/output control system that directs the processing of the data base.

8.10 Data Base Description Entry

an entry in the Schema, or Sub-schema, that describes a realm, set type or record type.

8.11 Data Base Exception Condition

the state that exists for a run unit when the DBCS detects a situation that requires special handling.

8.12 Data Base Key

a key whose value is assigned by the DBCS and which uniquely identifies a record occurrence within the data base.

8.13 Data Base Key Item

a data item whose contents represent the value of a data base key.

8.14 Data Base Status Indicator

a seven-digit integer value, consisting of a two-digit statement code and a five-digit status code, that indicates the state of the run unit with respect to the data base.

8.15 Data Clause

a clause appearing in the Record Statement of a Sub-schema that provides information describing a particular data item.

8.16 Data Item

the smallest unit of data defined by a Sub-schema and intended to be used as a FORTRAN variable.

8.17 Data Item Validator

the process of verifying the proper contents of data items as they are accessed from, or stored in, the data base.

8.18 Data Name

a user-defined word that names a data item described in a Data Description Clause.

8.19 Derived Data Item

A derived data item is a variable whose contents are generated by the DBCS, rather than by the FORTRAN program. There are two types of derived data items: actual derived data items and virtual derived data items. Derived data items are defined as such in the Schema.

An actual derived data item is one whose contents are generated by the DBCS and stored as a part of the data base record in which the data item is defined. The data item is a part of the record in the data base as long as the record exists in the data base. The DBCS generates the value of an actual derived data item during

execution of a STORE statement for the associated record, and during the execution of any statement that modifies the record or its relationships with other records. In addition, the DBCS modifies the data item whenever the data base is modified in any way that affects that data item's contents.

A virtual derived data item is one whose contents are generated by the DBCS during execution of a GET or FETCH statement for the record in which the data item is stored. A virtual derived data item does not exist in the data base, but exists only in the record area associated with the associated record type.

When a record that contains an actual derived data item is stored in the data base, the DBCS, using criteria defined in the Schema, generates the contents of the data item and stores it as a part of the data base record. The DBCS then ensures that the contents of the data item are modified whenever necessary to maintain valid contents, based on the criteria defined in the Schema.

There are three conditions under which the DBCS must modify the contents of an existing actual derived data item:

1. Whenever the record in which the data item is stored is modified in the data base, or
2. Whenever the record relationship of the record in which the data item is stored is modified in the data base, or
3. Whenever a different record or a different record's relationships are modified in the data base in a manner that affects the criteria for generating the contents of the actual derived data item.

Whenever the contents of an actual derived data item must be modified by the DBCS for any reason, the record in which the data item is stored must be in update usage mode.

8.20 Direct Location Mode

the location mode in which the user may specify to the DBCS a data base key value for use in the location mode algorithm.

8.21 Empty Set

a set containing no member records.

8.22 Fixed Set Order

a characteristic of a set type whose Set Description Entry specifies that the logical order of its member records is fixed permanently as defined in the Set Description Entry. A set with fixed set order, however, may be temporarily reordered during execution of a run unit, but the set's predefined fixed order is restored upon termination of that run unit.

8.23 Integrity

the ability to ensure that record relationships and the contents of data items are not adversely compromised by any process.

8.24 Invocation Privacy Lock

a privacy lock that must be satisfied in order to bind the Sub-schema definitions to a FORTRAN program.

8.25 Location Mode

the method by which the DBCS establishes the algorithm used to determine the data base key values associated with records.

8.26 Location Mode Data Item

a data item whose contents are used by the DBCS in evaluating a record's location mode or set selection criteria.

8.27 Locked Resource

a resource for which a privacy lock has been defined.

8.28 Manual Member

a record that explicitly becomes a member of a given set after the record has been stored in the data base by the use of the CONNECT statement.

8.29 Member Condition

the proposition, for which a truth value can be determined, that a set has members.

8.30 Null

a data attribute associated with currency indicators and data items defined in a Sub-schema. This attribute is independent of the value of the contents of data items. The representation of a null value is vendor-defined.

8.31 Object Sub-Schema

the machine-dependent representation that describes that portion of a data base known to a FORTRAN program.

8.32 Permanent Member

a record that, once it is a member of a given set, cannot be removed from membership in a set of that set type unless that record is removed from the data base.

8.33 Privacy Key

a key whose contents are used by the DBCS to determine if a locked resource is to be made available to the run unit that specified it.

8.34 Privacy Lock

a value known to the DBCS, that is associated with a locked resource.

8.35 Privacy Procedure

a set of statements used to generate a privacy key.

8.36 Protected Usage Mode

the state of a realm in which its records cannot be modified by another run unit.

8.37 Ready Mode

the state of a realm after successful execution of a READY statement and before the execution of a FINISH statement.

8.38 Realm

a logical subdivision of the data base which contains the records having a specified range of data base key values.

8.39 Realm Currency Indicator

the conceptual entity that is used by the DBCS as a place marker to indicate a specific record in a specific realm.

8.40 Realm Identifier

a data item associated with a given record type whose contents identify the realm in which that record is to be stored.

8.41 Realm Description Entry

an entry in the Schema or Sub-schema that defines a realm and its characteristics.

8.42 Realm Name

a user-defined word that names a realm described in a Realm Description Entry.

8.43 Record Selection Expression

Record Selection Expressions are specified in the FIND/FETCH Statements of a FORTRAN program for the purpose of selecting a specific record occurrence in the data base for subsequent use.

8.44 Record Type

the collection of record occurrences described by one Record Description Entry.

8.45 Record Type Currency Indicator

the conceptual entity that is used by the DBCS as a place marker to indicate a specific record occurrence of a specific record type.

8.46 Relation Record

a record, defined in a Sub-schema, which is used to define the relationship between records belonging to different sets.

8.47 Resource

a facility or service, controlled by the operating system, that can be used by an executing program.

8.48 Retrieval Usage Mode

the state of a realm during which its records may be accessed but cannot be modified.

8.49 Schema

the description of the entire data base. The Schema is specified by the Data Base Administration.

8.50 Schema Name

a representation that identifies a Schema.

8.51 Security

the ability to prohibit access to a data base record by unauthorized means.

8.52 Set

a logical collection of records in the data base. Each set consists of a set owner and zero or more set members.

8.53 Set Description Entry

an entry in the Schema or Sub-schema that defines a set type. The Schema also contains its characteristics.

8.54 Set Location Mode

the location mode in which the user may specify to the DBCS a set type for use in the location mode algorithm.

8.55 Set Member

a record stored in the data base as a non-owner participant in a specific set.

8.56 Set Name

a data-name that identifies all set occurrences of a given set type.

8.57 Set Ordering Criteria

the arguments used by the DBCS to determine the relative logical position of a record within a set.

8.58 Set Owner

a record in the data base whose occurrence establishes the existence of a specific set.

8.59 Set Selection Criteria

the arguments used by the DBCS to select one set occurrence from a collection of occurrences constituting a given set type.

8.60 Set Type

the collection of set occurrences described by a Set Description Entry.

8.61 Set Type Currency Indicator

the conceptual entity that is used by the DBCS as a place marker to indicate a specific record occurrence in a specific set occurrence of a specific set type.

8.62 Singular Set

a set for which the DBCS is the set owner.

8.63 Sub-schema

the named collection of statements that describes that portion of the data base accessible to a FORTRAN program.

8.64 Sub-Schema Description Entry

a statement that names the Sub-schema and describes some of its characteristics.

8.65 Sub-Schema Name

a representation that identifies a FORTRAN Sub-schema.

8.66 Statement Code

a value that indicates the type of statement whose execution has caused the DBCS to modify the Data Base Status Indicator. The statement code is a part of the Data Base Status Indicator.

8.67 Status Code

a value that indicates the absence or specific presence of Data Base Exception conditions that occurred during the most recent DML statement executed.

8.68 Temporary Realm

a realm that exists only during the execution of the run unit that creates it. When that run unit terminates, the temporary realms created by that run unit are logically removed from the data base.

8.69 Tenancy Condition

the proposition, for which a truth value can be established, to determine that a record is a member or owner of one or more sets.

8.70 Tenant

a record occurrence that is either the owner or a member of a specific set occurrence.

8.71 Transient Member

a record that can be removed from a given set.

8.72 Unsuccessful Execution

the attempted execution of a statement that does not result in the execution of all the operations specified by that statement. The unsuccessful execution of a statement does not affect any data referenced by that statement, but will affect status indicators.

8.73 Update Usage Mode

the state of a realm during which its records may be both accessed and modified.

8.74 Usage Mode

Usage mode designates the types of access that are permissible for a realm. The particular usage mode is specified in the READY statement.

Withdrawn/Retirée

APPENDIX A - SAMPLE SUB-SCHEMA AND FORTRAN PROGRAM

This section shows sets of sample programs. Each set consists of a Sub-schema definition followed by a short FORTRAN program.

In actual practice, it is felt that the complexity of the definition for a Sub-schema may dictate its preparation by a person who has expert knowledge of data base design principles (a Data Base Administrator).

Once the Sub-schema has been so prepared programmers will then be able to write their own FORTRAN applications(DML) programs.

```
C      THIS SUB-SCHEMA IS NAMED 'COUNT', IT REFERENCES
C      A SCHEMA WHICH IS NAMED 'ACCOUNTING-DATA'
C
C      SUBSCHEMA COUNT,SCHEMA='ACCOUNTING-DATA'
C
C      A USER OF THIS SUB-SCHEMA NEEDS TO REFERENCE DATA IN ONLY
C      TWO REALMS. NOTE THAT THERE MAY BE MORE THAN TWO REALMS
C      NAMED IN THE SCHEMA, AND THAT USERS OF THIS SUB-SCHEMA
C      CAN ACCESS ONLY THE TWO NAMED REALMS
C
C      REALM ACCT1,ACCT2
C
C      SIMILARLY, THE USER IS ALLOWED TO REFERENCE ONLY TWO SET
C      TYPES WHEN USING THIS SUB-SCHEMA.
C
C      SET ENG,MFG
C
C      THIS SUB-SCHEMA WILL NOT REQUIRE ANY ALIAS STATEMENTS.
C      THE NAMES SHOWN FOR DATA ITEMS, RECORDS, SETS AND REALMS
C      WERE CHOSEN TO BE ACCEPTABLE FORTRAN IDENTIFIERS.
C
C      THIS EXAMPLE SHOWS A SINGLE RECORD TO BE PROVIDED
C      TO THE FORTRAN USER. IT IS ASSUMED THAT THE SCHEMA
C      DECLARES THE RECORD TO BE OWNED DIRECTLY BY THE 'SYSTEM'
C
C      RECORD EMPLYE
C
C      THE RECORD IS NAMED 'EMPLYE', IT HAS A SINGLE ITEM OF
C      INTEREST TO THE USER OF THIS SUB-SCHEMA: SALARY.
C
C      REAL SALARY
C      END
```



```

C      THIS IS A SAMPLE APPLICATION PROGRAM.
C      IT USES SUB-SCHEMA #1, AND PRODUCES A COUNT OF THE
C      NUMBER OF EXEMPT EMPLOYEES.
C
C      PROGRAM PROG1
C
C      INVOKE(SUBSCHEMA='COUNT',SCHEMA='ACCOUNTING-DATA')
C      READY(ALL)
C
C      SET COUNTER ISUM TO ZERO
C
C      ISUM=0
C
C      IF THERE ARE NO RECORDS IN THE SET, THEN EXIT
C
C      IF(EMPTY(ENG)) GO TO 999
C
C      RETRIEVE THE INITIAL EMPLOYEE RECORD
C
C      FETCH(FIRST,RECORD=EMPLYE,SET=ENG,ERR=280)
100    ISUM=ISUM+1
C      FETCH(NEXT,RECORD=EMPLYE,SET=ENG,END=999,ERR=280)
C
C      IF END-OF-SET, THEN TO TO 999
C      IF ERROR, THEN GO TO 280
C
C      GO TO 100
C
C      280 PRINT 299,DBSTAT
C      299 FORMAT(' *ERROR IN RETRIEVAL* STATUS=',A)
C      FINISH(ALL)
C      STOP 1
C
C      999 PRINT 998,ISUM
C      998 FORMAT(' THERE ARE ',I10,' EXEMPT EMPLOYEES')
C
C      RELEASE THE REALM INVOKED
C
C      FINISH(ALL)
C      STOP 2
C      END

```


C THIS SUB-SCHEMA IS MORE COMPLEX THAN THE PREVIOUS EXAMPLE.
C IT USES THE ALIAS STATEMENT TO RENAME COBOL VARIABLES TO
C BE PROPER FORTRAN IDENTIFIERS, AND IT SPECIFIES THE USE
C OF DATA BASE PROCEDURES FOR LOGGING AND ERROR-MONITORING.
C

SUBSCHEMA COUNT2, SCHEMA='ACCOUNTING-DATA',
1 CALL NOTIFY BEFORE INVOKE,
2 CALL NOTIFY AFTER FINISH

C ASSUME THAT 'NOTIFY' IS A DATA BASE PROCEDURE WHICH RECORDS
C USE OF THIS SUBSCHEMA IN A SPECIAL FILE FOR LATER USE.
C

ALIAS(RECORD) EXEMPT='EXEMPT-STUDENT'

C RENAME EXEMPT-STUDENT TO BE EXEMPT. NOTE THAT THE NAME
C EXEMPT-STUDENT NOW BECOMES UNAVAILABLE FOR USE IN EITHER
C THE FOLLOWING SUB-SCHEMA STATEMENTS, OR THE USER'S FORTRAN
C PROGRAM. THAT RECORD MUST BE REFERENCED BY THE ALIAS
C EXEMPT.
C

REALM ACCT1
SET ENG

C RECORD EXEMPT(CHECK=DATA TYPE, CALL LOGIT BEFORE FETCH)

C THE RE-NAMED RECORD IS TO HAVE ITS CONTENTS VERIFIED BY
C INDIVIDUAL DATA TYPE CHECKING DURING "TRANSIT" TO AND
C FROM THE DATA BASE. A DATA BASE PROCEDURE LOGIT, WHICH
C CAN BE ASSUMED SIMILAR IN FUNCTION TO THE PREVIOUS
C NOTIFY, IS AUTOMATICALLY INVOKED BY THE DBCS PRIOR TO EACH
C FETCH STATEMENT EXECUTION.
C

REAL SALARY
END


```

C      THIS PROGRAM DETERMINES THE NUMBER OF EXEMPT STUDENTS
C      EMPLOYED BY AN ENGINEERING DEPARTMENT, AND COMPUTES THE
C      TOTAL PAYROLL PAID TO THOSE STUDENTS DURING A ONE YEAR
C      PERIOD.
C
C      PROGRAM PROG2
C      INVOKE(SUBSCHEMA='COUNT2',SCHEMA='ACCOUNTING-DATA')
C      ISUM=0
C
C      ISUM COUNTS PEOPLE
C
C      SUMSAL=0
C
C      SUMSAL COUNTS TOTAL SALARY
C
C      READY(REALM=ACCT1)
C
C      GO TO 999 IF NO RECORDS IN THE SET
C
C      IF(EMPTY(ENG)) GO TO 999
C
C      FETCH THE FIRST RECORD
C
C      FETCH(FIRST,RECORD=EXEMPT,SET=ENG,ERR=888)
100    ISUM=ISUM+1
      SUMSAL=SUMSAL+SALARY
      FETCH(NEXT,RECORD=EXEMPT,SET=ENG,END=999,ERR=888)
      GO TO 100
C
C      LOOP TILL END-OF-SET OR ERROR
C
C      888 PRINT 887,DBSTAT
C      887 FORMAT(' ERROR CODE= ',A,' PROGRAM TERMINATED')
C      FINISH(ALL)
C      STOP1
C
C      999 PRINT 998,ISUM,SUMSAL
C      998 FORMAT(' ',I7,' STUDENTS RECEIVED A TOTAL OF $',F10.2)
C      FINISH(ALL)
C
C      CLOSE THE DATA BASE
C
C      END

```


APPENDIX B - FORTRAN EXTENSIONS NOT PRESENTLY IN COBOL DATA BASE FACILITIES

The FORTRAN Data Base facility contains several new features over the COBOL facility [6]. They are as follows:

1. The FETCH DML statement - See Section 4.5
2. Two new membership conditions FIRST and LAST - See Section 2.6
3. A <db name> phrase can be a character expression (i.e. dynamically interpreted at run-time) - See Section 2.3.6
4. Record selection expressions can be a character expression - See Section 4.6
5. Individual DML statements can specify the action to be taken if a data base exception occurs - See Section 2.3.10 and 5.2
6. A FIND or FETCH DML statement can specify what action to be taken if the end of set or end of realm data base exception occurs - See Section 2.3.9 and 5.2
7. The USE functionality has been divided into two statements, USE for data base exceptions and PRIVACY for privacy key processing. Additionally these two statements are now procedural rather than declarative, i.e. the set of USE/PRIVACY conditions can be changed at run-time. - See Section 4.12 and 4.15
8. The keyword ALL is used in the Sub-schema lock specifications to indicate that privacy locks are to be established for all possible FORTRAN DML statements.
9. If an empty lock list specification is used in the Sub-schema lock specification, a lock list is established by default for the DML statements which provide minimum access to the DBCS.

INDEX

THIS INDEX HAS DEFINITIONS FIRST FOLLOWED BY REFERENCES.
NOTE THAT THE REFERENCES ARE TO SECTION NUMBERS.

Abnormal Termination 8.1
 ACCEPT 4.1, 5.3, 6.46
 Accept, Currency 4.1.1, 6.47
 Accept Currency -See Accept, Currency
 Accept, Realm Name 4.1.2, 6.48
 Accept Realm Name -See Accept, Realm Name
 ACCEPT STATEMENT 4.1, 6.46
 Access, Direct 2.4.1, 6.30
 Access, Indicator, Currency 2.4.5, 6.34
 Access, Mode, Calculate 2.4.2, 6.31
 Access, Positional 2.4.4, 6.33
 Access, Privacy, Item 4.12.4, 6.64
 Access, Privacy, Realm 4.12.2, 6.61
 Access, Privacy, Record 4.12.3, 6.62
 Access, Privacy, Set 4.12.3.1, 6.63
 Access, Privacy, Sub-schema 4.12.1, 6.60, 4.12, 6.59, 6.60
 Access, Set, Owner 2.4.6, 6.35
 Access, Set, Search 2.4.3, 6.32
 Access, Set, Selection Rules 2.4.7, 6.36
 ALIAS 2.3.6, 3, 3.2, 8.2, 8.74, 6.41
 Alias Description Entry 8.2
 ALIAS STATEMENT 3.2, 8.74
 ALL 3.1, 3.3, 3.4, 3.5, 3.6, 4.2, 4.3, 4.4, 4.7, 4.10,
 4.12.3.1, 4.13, 4.15, 8.74, 6.40, 6.42, 6.43, 6.44, 6.45,
 6.54, 6.57, 6.63
 <argument> 2.3.1, 6.1, 2.3.8, 2.3.9, 2.3.10, 6.8, 6.9, 6.10
 Arguments 2.3.1, 6.1
 <array declarator> 3.6, 6.45
 ASCENDING 4.11
 AUTOMATIC 2.1.4.3.3
 Automatic Member -See Member, Automatic
 Available Mode -See Mode, Available
 Braces 2.2
 Brackets 2.2
 <calc entry> 3.4, 6.43
 Calculate Mode Access -See Access, Mode, Calculate
 Calculation Key -See Key, Calculation
 Calculation Location Mode -See Mode, Location, Calculation
 Character Constant -See Constant, Character
 Character Expression -See Expression, Character
 Character Variable 2.3.4, 6.4
 <char con> 2.3.2, 6.2, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 4.9,
 4.12.1, 4.12.2, 4.12.3, 4.12.3.1, 4.12.4, 6.40, 6.41, 6.42,
 6.43, 6.44, 6.45, 6.56, 6.60, 6.61, 6.62, 6.63, 6.64
 <char exp> 2.3.3, 6.3, 2.3.6, 4.5, 4.6, 4.12, 6.6, 6.52,
 6.53, 6.59
 <char var> 2.3.4, 6.4, 3.5, 4.1.2, 6.44, 6.48
 <check option> 3.5, 6.44
 Clause, Data 8.15
 CODASYL 7

CODASYL Data Base Facility -See CODASYL
 Code, Statement 8.66, 5.1.1, 5.3
 Code, Status 8.67, 5.1.1, 5.3
 Concepts, Data Base 2, 2.1
 CONCURRENT 2.3.28, 4.13, 6.28
 Concurrent Run Unit -See Run-unit, Concurrent
 Condition, Member 2.6, 8.29, 6.38, 8.29
 Condition, Null 2.7, 6.39
 Condition, Tenancy 2.5, 8.69, 6.37
 Condition, USE, Processing of 5.2.1
 CONNECT 2.1.4.3.3, 4.2, 4.12.3.1, 5.3, 6.49, 6.63
 CONNECT STATEMENT 4.2, 6.49
 Constant, Character 2.3.2, 6.2
 Currency Indicator Access -See Access, Indicator, Currency
 Currency Indicator -See Indicator, Currency
 Currency Indicators -See Indicator, Currency
 <currency type> 4.1.1, 6.47
 DATA BASE CONCEPTS -See Concepts, Data Base
 Data Base Control System (DBCS) 2.1.2.1, 8.9, 2.1.2, 2.1.3,
 2.1.4, 2.1.4.1, 2.1.4.3, 2.1.4.4.1, 2.1.4.4.2, 2.1.4.4.3,
 2.1.4.5, 2.3.5, 2.3.6, 2.3.7, 2.3.9, 2.3.10, 2.3.12, 2.4,
 2.4.3, 2.4.7, 3.1, 3.4, 3.5, 4.8, 4.12, 4.12.2, 4.14, 5,
 5.1, 5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.2, 5.2.1, 5.2.2, 5.3,
 8.4, 8.5, 8.6, 8.8, 8.9, 8.11, 8.12, 8.19, 8.20, 8.25,
 8.26, 8.33, 8.39, 8.45, 8.54, 8.57, 8.59, 8.61, 8.62, 8.66,
 8.74
 Data Base Description Entry -See Entry, Description
 Data Base Environment -See Environment, Data Base
 Data Base Exception Condition -See Exception
 Data Base Exception -See Exception
 Data Base Facility 2.1.1, 2.2
 Data Base Key Item -See Key Item, Data Base
 Data Base Key -See Key, Data Base
 Data Base Keys -See Key, Data Base
 Data Base Name -See Name, Data Base
 Data Base Names -See Name, Data Base
 Data Base Procedure -See Procedure, Data Base
 Data Base Procedures -See Procedure, Data Base
 Data Base Registers -See Register, Data Base
 Data Base Special Registers -See Register, Data Base
 Data Base Status Indicator -See Indicator, Data Base
 Data Base Status Indicators -See Indicator, Data Base
 Data Base Task Group -See CODASYL
 Data Clause -See Clause, Data
 Data Item -See Item, Data
 Data Item Validator -See Item, Data, Validator
 Data Manipulation Language (DML) 4, 1, 2.2, 4.12.3, 8.74
 Data Name -See Name, Data
 DBCS -See Data Base Control System (DBCS)
 <db key> 2.3.5, 6.5, 2.4.1, 3.4, 4.1.1, 4.1.2, 6.30, 6.43,
 6.47, 6.48
 <db name> 2.3.6, 6.6, 2.2, 2.3.7, 2.3.18, 2.3.19, 2.3.20,
 2.3.23, 2.4.1, 2.4.2, 2.4.3, 2.4.4, 2.4.5, 2.4.6, 2.4.7,
 2.5, 2.6, 3.3, 3.4, 3.5, 4.1.1, 4.1.2, 4.2, 4.3, 4.4, 4.8,
 4.10, 4.11, 4.14, 8.74, 6.6, 6.7, 6.18, 6.19, 6.23, 6.30,
 6.31, 6.32, 6.33, 6.34, 6.35, 6.36, 6.37, 6.38, 6.42, 6.43,
 6.44, 6.47, 6.48, 6.49, 6.50, 6.51, 6.55, 6.57, 6.58, 6.66
 <db names> 2.3.7, 6.7, 2.2, 2.3.20, 3.5, 4.2, 4.3, 4.4, 4.7,

4.10, 4.11, 4.12.2, 4.12.3, 4.12.3.1, 4.12.4, 4.13, 6.20,
 6.44, 6.49, 6.50, 6.51, 6.54, 6.57, 6.58, 6.61, 6.62, 6.63,
 6.64, 6.65
 DBPRIV - Data Base Privacy Register -See Register, Data Base
 <db proc> 2.3.8, 6.8, 3.1, 3.3, 3.4, 3.5, 3.6, 6.40, 6.42,
 6.43, 6.44, 6.45
 DBREC - Data Base Record Register -See Register, Data Base
 DBRELM - Data Base Realm Register -See Register, Data Base
 DBSET - Data Base Set Register -See Register, Data Base
 DBSTAT - Data Base Status Register -See Register, Data Base
 Derived Value -See Value, Derived
 Direct Access -See Access, Direct
 Direct Location Mode -See Mode, Location, Direct
 DISCONNECT 2.1.4.3.4, 3.5, 4.3, 4.4, 4.12.3.1, 5.3, 6.44,
 6.50, 6.53
 DISCONNECT STATEMENT 4.3, 6.50
 DML language -See Data Manipulation Language (DML)
 DML statements -See Data Manipulation Language (DML)
 EMPTY 2.6
 Empty Set -See Set, Empty
 <end> 2.3.9, 6.9, 4.5, 4.6, 5.2.2, 6.52, 6.53
 END Phrase 2.3.9, 6.9
 END STATEMENT 3.7
 Entry, Description 8.10
 Entry, Description, Realm 8.41
 Entry, Description, Set 8.53, 4.11, 8.22
 Entry, Description, Sub-schema 8.64
 <entry specifier> 3.4, 6.43
 Environment, Data Base 2.1.2
 <erase> 4.12.3, 4.12.3.1, 6.62, 6.63, 4.12.3.1, 6.62, 6.63
 ERASE 2.1.4.3.4, 2.2, 3.5, 4.4, 4.12.3, 4.12.3.1, 5.3, 6.44,
 6.51, 6.62, 6.63
 ERASE STATEMENT 4.4, 6.51
 <error> 2.3.10, 6.10, 4.1.1, 4.1.2, 4.2, 4.3, 4.4, 4.5, 4.6,
 4.7, 4.8, 4.10, 4.11, 4.13, 4.14, 5, 5.2, 5.2.1, 5.2.2,
 6.47, 6.48, 6.49, 6.50, 6.51, 6.52, 6.53, 6.54, 6.55, 6.57,
 6.58, 6.65, 6.66
 Error Codes -See Exception
 Error Phrase 2.3.10, 6.10
 Exception 5, 5.2, 8.11, 8.72, 5.1.1, 5.1.2, 5.1.3, 5.1.4,
 5.2, 5.2.1, 5.3, 8.11
 EXCEPTION HANDLING -See Exception
 EXCLUSIVE 2.3.28, 3.3, 4.11, 4.13, 6.42
 Expression, Character 2.3.3, 6.3
 Expression, Selection, Record 2.1.4.4.4, 8.43
 FETCH 2.1.4.4.4, 2.4, 3.4, 4.5, 5.3, 8.19, 8.74, 6.52
 FETCH STATEMENT 4.5, 8.74
 FIND 2.3.9, 2.3.20, 2.4, 3.4, 3.5, 4.5, 4.6, 4.12.3.1, 5.3,
 8.74, 6.43, 6.53, 6.63
 FIND STATEMENT 4.6, 6.53
 FINISH 2.3.7, 4.7, 4.12, 4.13, 5.3, 8.37, 6.54
 FINISH STATEMENT 4.7
 FIRST 2.4.4, 2.4.7, 2.6, 3.4, 8.74, 6.36, 6.43
 Fixed Set Order -See Set, Fixed, Order
 FORTRAN SUB-SCHEMA -See Sub-schema
 GET STATEMENT 4.8, 6.55
 GLOSSARY 8
 <identifier> 2.3.11, 6.11, 2.3.12, 2.3.22, 2.3.27, 2.7, 3.2,

3.4, 3.6, 4.15, 6.12, 6.22, 6.27, 6.39, 6.41, 6.43, 6.45, 6.67
 Identifier 2.3.11, 6.11, 2.3.12, 2.3.13, 2.3.14, 6.12, 6.13, 6.14
 Identifier Item 2.3.12, 6.12
 Identifier List -See List, Identifier
 Identifier, Realm 8.40
 Identifier Specification -See Specification, Identifier
 <id item> 2.3.12, 3.4, 6.43
 <id list> 2.3.13, 6.13, 2.4.3, 2.4.7, 4.10, 4.11, 4.12.4, 6.32, 6.36, 6.57, 6.58, 6.64
 <id spec> 2.3.14, 6.14, 2.3.13, 6.13
 Indicator, Currency 2.1.4.4.1, 8.8, 2.4, 2.4.5, 5.3, 8.8, 6.29, 6.34
 Indicator, Currency, Realm 8.39
 Indicator, Currency, Record 8.45
 Indicator, Currency, Set 8.61
 Indicator, Data Base 5.3, 8.14, 5.1.1, 8.14
 <integer> 2.3.15, 6.15, 2.2
 Integer Constant 2.3.15, 6.15
 Integrity 8.23
 <int exp> 2.4.4, 6.33
 Invocation Privacy Lock -See Lock, Privacy, Invocation
 INVOKE 2.1.3, 3.1, 4.9, 6.56
 INVOKE STATEMENT 4.9
 ITEM 8.74
 Item, Data 8.16, 8.19, 5.3, 8.19
 Item, Data, Validator 8.17
 <item dml stmt> 4.12.4, 6.64
 <item list> 2.3.16, 6.16, 4.5, 4.8, 6.52, 6.55
 Item List -See List, Item
 <item spec> 2.3.17, 6.17, 2.3.16, 6.16
 Item Specification -See Specification, Item
 Key, Calculation 8.5
 Key, Data Base 2.1.4.1, 2.3.5, 8.12, 6.5, 8.12
 <key item> 3.4, 6.43
 Key Item, Data Base 8.13
 Key Item -See Item, Key
 <key list> 3.4
 Key, Privacy 8.33
 LAST 2.4.4, 2.4.7, 2.6, 3.4, 8.74, 6.36, 6.38, 6.43
 List, Identifier 2.3.13, 6.13
 List, Item 2.3.16, 6.16
 List, Status 2.3.25, 6.25
 LOCALLY 4.11
 Location Mode Data Item -See Mode, Location, Data Item
 Location Mode -See Mode, Location
 Location Modes -See Mode, Location
 Locked Resource -See Lock, Resource
 <lock list> 3.1, 6.40
 Lock, Privacy 8.34, 5.3
 Lock, Privacy, Invocation 8.24
 Lock, Resource 8.27
 <lock spec> 3.1, 3.3, 3.4, 3.5, 3.6, 6.40, 6.42, 6.43, 6.44, 6.45, 3.3, 3.4, 3.5, 3.6, 6.40, 6.42, 6.43, 6.44, 6.45
 MANUAL 4.2, 4.3, 4.4
 Manual Member -See Member, Manual
 MEMBER 2.5

Member, Automatic 8.3
 <member condition> 2.6, 6.38
 Member Condition -See Condition, Member
 Member, Manual 8.28
 Member, Permanent 8.32
 Member, Set 2.1.4.3.2, 8.55
 Membership, Class, Removal of 2.1.4.3.4
 MEMBERSHIP CONDITIONS -See Condition, Member
 Membership, Storage Class 2.1.4.3.3
 Member, Transient 8.71
 <member type> 4.4, 6.51
 Meta-Language 2.2, 2.3
 <meta-symbol-name> 2.2
 META-SYMBOLS -See Meta-Language
 Mode, Available 8.4
 Mode, Location 2.1.4.4.2, 8.25
 Mode, Location, Calculation 8.6
 Mode, Location, Data Item 8.26
 Mode, Location, Direct 8.20
 Mode, Location, Set 8.54
 Mode, Ready 8.37, 5.3
 Mode, Usage 8.74
 Mode, Usage, Protected 8.36
 Mode, Usage, Retrieval 8.48
 Mode, Usage, Update 8.73
 MODIFY 3.4, 3.6, 4.10, 4.12.4, 5.3, 6.57, 6.64
 MODIFY STATEMENT 4.10, 6.57
 Name, Data 8.18
 Name, Data Base 2.3.6, 2.3.7, 6.6, 6.7
 <name in schema> 3.2, 6.41
 Name, Realm 2.3.18, 8.42, 6.18, 8.42
 Name, Record 2.3.19, 6.19
 Name, Schema 2.3.22, 8.50, 6.22, 8.50
 Name, Set 2.3.23, 8.56, 6.23, 8.56
 Name, Subroutine 2.3.26, 6.26
 Name, Sub-schema 8.65
 <name type> 4.1.2, 6.48
 NOTATIONAL RULES -See Meta-Language
 Null 8.30
 NULL 2.7, 6.39
 <null cond> 2.7
 NULLITY CONDITION -See Condition, Null
 Number, Statement 2.3.24, 6.24
 Object Sub-Schema -See Sub-schema, Object
 ORDER 4.11, 4.12.3.1, 5.3, 6.58, 6.63
 Order, Set 2.1.4.3.1, 8.57
 ORDER STATEMENT 4.11, 6.58
 OWNER 2.5, 3.4, 6.37, 6.43
 Owner, Set 8.58, 2.4, 2.4.6, 6.29, 6.35
 <path specifier> 3.4, 6.43
 PERMANENT 2.2, 4.4, 4.12.3.1, 6.63
 Permanent Member -See Member, Permanent
 <posit> 2.4.4, 6.33
 Positional Access -See Access, Positional
 PREFACE 1
 Primary Key -See Key, Primary
 PRIVACY 3.1, 3.3, 3.4, 3.5, 3.6, 4.9, 4.12, 4.12.4, 4.13,
 5.1.5, 5.3, 8.74, 6.40, 6.42, 6.43, 6.44, 6.59

Privacy Access for Item(s) -See Access, Privacy, Item
 Privacy Access for Realm -See Access, Privacy, Realm
 Privacy Access for Record(s) -See Access, Privacy, Record
 Privacy Access for Set(s) -See Access, Privacy, Set
 Privacy Access for Sub-schema -See Access, Privacy, Sub-schema
 Privacy Display Access for Sub-schema -See Access, Privacy
 Privacy Key -See Key, Privacy
 Privacy Lock -See Lock, Privacy
 Privacy Procedure -See Procedure, Privacy
 PRIVACY STATEMENT 4.12, 4.12.4, 6.64
 <priv disable> 4.12, 6.59
 <priv enable> 4.12
 <priv key> 4.12, 4.12.1, 4.12.2, 4.12.3, 4.12.3.1, 4.12.4,
 6.60, 6.61, 6.62, 6.63, 6.64
 Procedural Flow After Data Base Exceptio-See Exception
 Procedural Flow -See Exception
 Procedure, Data Base 2.1.4.5, 2.3.8, 6.8
 Procedure, Privacy 8.35
 Processing an <error> or <end> Phrase 5.2.2
 Processing A USE Condition -See Condition, USE, Processing of
 PROTECTED 2.3.28, 3.3, 4.11
 Protected Usage Mode -See Mode, Usage, Protected
 READY 2.3.28, 3.3, 4.12, 4.12.2, 4.13, 5.3, 8.37, 8.74, 6.65
 Ready Mode -See Mode, Ready
 READY STATEMENT 4.13
 Realm 2.1.4.2, 8.38, 2.3.18, 2.3.20, 3.5, 4.1.2, 5.1.2, 5.3,
 8.38, 8.42, 6.18, 6.20, 6.48
 REALM 2.2, 2.3.18, 2.3.20, 3, 3.3, 3.5, 4.1.2, 4.12.2, 8.74,
 6.18, 6.20, 6.42, 6.48
 <realm control> 3.3, 6.42
 Realm Currency Indicator -See Indicator, Currency, Realm
 <realm declaration> 3.3
 Realm Description Entry -See Entry, Description, Realm
 Realm Identifier -See Identifier, Realm
 <realm-id option> 3.5, 6.44
 <realm lock> 3.3, 6.42
 <realm lock list> 3.3
 <realm lock option> 3.3
 <realm lock type> 3.3
 <realm name> 2.3.18, 6.18, 2.3.6, 3.2, 3.3, 6.6, 6.41
 Realm Name -See Name, Realm
 Realms -See Realm
 REALM STATEMENT 3.3, 6.42
 Realm, Temporary 8.68
 <rec dml stmt> 4.12.3, 6.62
 RECORD 2.3.19, 2.3.20, 2.4, 2.4.7, 3, 3.2, 3.5, 4.2, 4.11,
 8.74, 6.19, 6.20, 6.29, 6.36, 6.44, 6.49
 <record control> 3.5, 6.44
 <record declaration> 3.5
 <record lock list> 3.5
 <record lock option> 3.5
 <record lock type> 3.5
 <record name> 2.3.19, 6.19, 2.3.6, 3.2, 6.6, 6.41
 Record Name -See Name, Record
 <record option> 3.5, 6.44
 Record Placement and Selection -See Selection, Record
 Record, Relation 8.46
 Records 2.1.4

RECORD SELECTION EXPRESSIONS 2.4, 6.29
 Record Selection Expression -See Expression, Selection, Record
 Records in the Data Base -See Records
 RECORD STATEMENT 3.5, 6.44
 Record Type Currency Indicator -See Indicator, Currency, Record
 Record Type -See Type, Record
 REFERENCES 7, 8.74
 Register, Data Base 5.1, 5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5
 Relation Record -See Record, Relation
 Removal Class of Membership -See Membership, Class, Removal of
 <re-name> 3.2, 6.41
 Repetition 2.2
 Resource 8.47, 5.3
 <retain> 2.2, 2.3.20, 6.20, 2.3.20, 4.5, 4.6, 4.14, 6.52, 6.53, 6.66
 Retaining Clause 2.3.20, 6.20
 <retain-1> 2.2
 <retain1> 2.3.20
 <retain-2> 2.2
 <retain2> 2.3.20
 RETRIEVAL 2.3.28, 3.3, 4.11, 4.13, 6.42, 6.65
 Retrieval Usage Mode -See Mode, Usage, Retrieval
 <rse> 2.4, 6.29, 4.5, 4.6, 6.52, 6.53
 <rse 1> 2.4.1, 6.30, 2.4, 6.29
 <rse 2> 2.4.2, 6.31, 2.4, 6.29
 <rse 3> 2.4.3, 6.32, 2.4, 6.29
 <rse 4> 2.4.4, 6.33, 2.4, 6.29
 <rse 5> 2.4.5, 6.34, 2.4, 6.29
 <rse 6> 2.4.6, 6.35, 2.4, 6.29
 <rse 7> 2.4.7, 6.36, 2.4, 6.29
 RUNUNIT 4.1.1, 4.1.2
 Run-unit, Concurrent 8.7
 Schema 2.1.2.2, 8.49, 2.1.2, 2.1.3, 2.1.4.3.1, 2.1.4.4.2, 2.1.4.5, 2.3.6, 2.3.21, 2.3.22, 3.1, 3.2, 3.4, 3.5, 3.6, 4.12, 4.14, 8.6, 8.41, 8.49, 8.53, 6.21, 6.22
 Schema and Sub-Schema, Relationship 2.1.3
 <schema item name> 2.3.21, 6.21, 3.2, 6.41
 Schema Item Name 2.3.21, 6.21
 <schema name> 2.3.22, 6.22, 3.1, 4.9, 6.40, 6.56
 Schema Name -See Name, Schema
 <schema name type> 3.2, 6.41
 Security 8.51, 1
 <selection option> 3.4, 6.43
 Selection, Record 2.1.4.4
 Selection, Set 2.1.4.4.3, 8.59
 SELECTIVE 4.4, 4.12.3.1, 6.63
 Set 2.1.4.3, 8.52, 2.1.3, 2.1.4.3.1, 2.1.4.3.2, 2.1.4.4.3, 2.3.23, 2.4, 2.4.3, 5.1.3, 5.3, 8.22, 8.52, 8.60, 6.23, 6.29, 6.32
 SET 2.1.4.4.3, 2.3.20, 2.3.23, 2.4.3, 3, 3.4, 5.1.3, 8.74, 6.23, 6.43
 <set control> 3.4
 <set declaration> 3.4
 Set Description Entry -See Entry, Description, Set
 <set dml stmt> 4.12.3.1, 6.63
 Set, Empty 8.21
 Set, Fixed, Order 8.22

Set Location Mode -See Mode, Location, Set
 <set lock list> 3.4, 6.43
 <set lock option> 3.4
 <set lock type> 3.4
 Set Member -See Member, Set
 Set Membership -See Member, Set
 <set name> 2.3.23, 6.23, 2.3.6, 3.2, 6.6, 6.41
 Set Name -See Name, Set
 Set Occurrence Selection Rules Access -See Access, Set, Selection
 Set Occurrence Selection Rule -See Access, Set, Selection
 <set option> 3.4, 6.43
 Set Ordering Criteria -See Order, Set
 Set Ordering -See Order, Set
 Set Owner Access -See Access, Set, Owner
 Set Owner -See Owner, Set
 Set Search Access -See Access, Set, Search
 Set Selection Criteria -See Selection, Set
 Set Selection -See Selection, Set
 Set, Singular 8.62
 Sets, Set Types, and Set Occurrences -See Set
 SET STATEMENT 3.4
 Set Type Currency Indicator -See Indicator, Currency, Set
 Set Type -See Type, Set
 Singular Set -See Set, Singular
 <size constant> 3.6, 6.45
 <sort field> 4.11, 6.58
 <sort spec> 4.11
 Specification, Identifier 2.3.14, 6.14
 Specification, Item 2.3.17, 6.17
 Statement Code -See Code, Statement
 <statement number> 2.3.24, 6.24, 2.3.9, 2.3.10, 6.9, 6.10
 Statement Number -See Number, Statement
 STATEMENT SYNTAX SUMMARY -See Syntax
 STATUS 2.2, 5.3
 Status Code -See Code, Status
 Status Indicator -See Indicator, Status
 <status list> 2.3.25, 6.25, 4.15, 6.67
 Status List -See List, Status
 Storage Class of Membership -See Membership, Storage Class
 STORE STATEMENT 4.14, 6.66
 <subroutine name> 2.3.26, 6.26, 2.3.8, 2.3.9, 2.3.10, 4.12, 6.8, 6.9, 6.10, 6.59
 Subroutine Name -See Name, Subroutine
 Sub-schema 2.1.2.3, 3, 8.63, 2.1.2, 2.1.3, 2.1.4, 2.2, 2.3.6, 2.3.19, 2.3.23, 2.3.27, 2.4.3, 3, 3.1, 3.2, 3.4, 3.5, 3.6, 3.7, 4.9, 4.11, 4.13, 5.2, 8.2, 8.15, 8.16, 8.41, 8.53, 8.63, 8.64, 8.74, 6.19, 6.23, 6.27
 SUBSCHEMA 2.1.3, 3, 3.1, 4.9, 4.12.1, 8.74, 6.40, 6.56
 <subschema control> 3.1, 6.40
 Sub-Schema Description Entry -See Entry, Description, Sub-schema
 <subschema key option> 3.1
 <subschema lock option> 3.1
 <subschema lock type> 3.1
 <subschema name> 2.3.27, 6.27, 3.1, 4.9, 4.12.2, 4.12.3, 4.12.3.1, 4.12.4, 6.56, 6.61, 6.62, 6.63, 6.64
 Sub-schema Name 2.3.27, 6.27

Sub-Schema Name -See Name, Sub-schema
 Sub-schema, Object 8.31
 SUBSCHEMA STATEMENT 3.1, 6.40
 Syntax 6, 2.2, 5.3
 Syntax Expression -See Syntax
 Syntax Rules -See Syntax
 Temporary REALM -See Realm, Temporary
 <tenancy condition> 2.5, 6.37
 Tenancy Condition -See Condition, Tenancy
 Tenant 8.70
 TENANT 2.5
 TRANSIENT 2.1.4.3.4
 Transient Member -See Member, Transient
 TYPE 8.74
 <type declaration> 3.6, 6.45
 Type, Record 8.44, 8.45
 Type, Set 8.60
 <type specification> 3.6, 6.45
 TYPE-STATEMENTS 3.6
 Unsuccessful Execution -See Exception
 Update Usage Mode -See Mode, Usage, Update
 <usage> 2.3.28, 6.28, 4.12.2, 4.13, 6.61, 6.65
 Usage 2.3.28, 6.28, 5.3, 8.73, 8.74
 Usage Mode -See Mode, Usage
 USE 2.3.9, 2.3.10, 4.15, 5, 5.2, 5.2.1, 5.2.2, 5.3, 8.74,
 6.67
 USE STATEMENT 4.15
 <variable lock list> 3.6, 6.45, 3.4, 6.45
 <variable lock type> 3.6
 <variable name> 3.4
 <variable option> 3.6
 <within option> 3.5, 6.44