TC 97

INTERNATIONAL STANDARD

ISO
8907

First edition
1987-06-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ

Information processing systems — Database language NDL

Reference number
ISO 8907 : 1987 (E)

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75% approval by the member bodies voting.

International Standard ISO 8907 was prepared by Technical Committee ISO/TC 97, *Information processing systems*.

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

Printed in Switzerland

Contents

1. Scope and field of application **1**

2. References **3**

3. Overview **5**
3.1 Organization 5
3.2 Notation 5
3.3 Conventions 6
3.4 Conformance 7

4. Concepts **9**
4.1 Data type 9
4.1.1 Character strings 9
4.1.2 Numbers 9
4.2 Components 10
4.3 Records 10
4.3.1 Database keys 10
4.3.2 System record 11
4.4 Set types 11
4.4.1 Singular sets 11
4.4.2 Recursive sets 11
4.5 Schema 11
4.6 Subschemas 12
4.7 Database 12
4.8 Modules 12
4.9 Procedures 12
4.10 Standard programming languages 13
4.11 Temporary sets 13
4.12 Parameters 13
4.12.1 STATUS parameter 13
4.12.2 TEST parameter 13
4.12.3 RECORD parameter 13
4.13 Session 14
4.14 Transactions 14
4.15 Integrity constraints 14
4.15.1 Check conditions 15
4.15.2 Default values 15
4.15.3 Uniqueness constraints 15
4.15.4 Set ordering criteria 15
4.15.5 Set membership 15

5. Common elements **17**
5.1 <condition> 17
5.2 <operand> 19
5.3 <identifier> 20
5.4 <literal> 22
5.5 <data type> 24
5.6 <occurs clause> 26
5.7 <subscripts> 27

5.8 Comments, spaces, and key words 28

6. Schema definition language 31
6.1 <schema> 31
6.2 <schema name clause> 32
6.3 <record type> 33
6.4 <record name clause> 34
6.5 <record uniqueness clause> 35
6.6 <component type> 36
6.7 <component name clause> 37
6.8 <default clause> 38
6.9 <record check clause> 39
6.10 <set type> 40
6.11 <set name clause> 41
6.12 <owner clause> 42
6.13 <order clause> 43
6.14 <member clause> 45
6.15 <member record name clause> 46
6.16 <insertion clause> 47
6.17 <retention clause> 49
6.18 <member uniqueness clause> 50
6.19 <key clause> 51
6.20 <member check clause> 54
6.21 <component identifier> 56

7. Subschema definition language 59
7.1 <subschema> 59
7.2 <subschema name clause> 60
7.3 <record view> 61
7.4 <component view> 62
7.5 <set view> 63

8. Module language 65
8.1 <module> 65
8.2 <module name clause> 67
8.3 <temporary set specifications> 68
8.4 <procedure> 69

9. Data manipulation language 75
9.1 <commit statement> 75
9.2 <connect statement> 76
9.3 <disconnect statement> 77
9.4 <erase statement> 78
9.5 <find statement> 80
9.6 <get statement> 85
9.7 <modify statement> 86
9.8 <nullify cursor statement> 88
9.9 <ready statement> 89
9.10 <reconnect statement> 91
9.11 <rollback statement> 92
9.12 <store statement> 93
9.13 <test database key equal statement> 96
9.14 <test database key null statement> 97
9.15 <test set empty statement> 98
9.16 <test set membership statement> 99
9.17 <database key identifier> 100

9.18 <component view identifier> 101
9.19 <parameter identifier> 103
9.20 <to parameter move> and <to database move> 104

10. Auxiliary operations **107**
10.1 <insert operation> 107
10.2 <remove operation> 109

11. Interpretive state **111**
11.1 <session state> 111
11.2 <cursors> 112
11.3 <temporary sets> 114
11.4 <ready list> 115

12. Status codes **117**

13. Levels **119**

Annexes **121**

A. Example suppliers and parts problem **121**
A.1 Example suppliers and parts <schema> 121
A.2 Example suppliers and parts <subschema> 122
A.3 Example suppliers and parts program 122
A.4 Example suppliers and parts <module> 123

B. Example recursive set problem **125**
B.1 Example recursive set <schema> 125
B.2 Example recursive set <subschema> 125
B.3 Example recursive set program 125
B.4 Example recursive set <module> 127

C. Example bill of materials problem **129**
C.1 Example bill of materials <schema> 129
C.2 Example bill of materials <subschema> 130
C.3 Example bill of materials program 130
C.4 Example bill of materials <module> 131

Index **133**

Information processing systems — Database language NDL

1. Scope and field of application

This standard specifies the syntax and semantics of three database languages:

1) A schema definition language, for declaring the structures and integrity constraints of an NDL database.

2) A subschema definition language, for declaring a user view of that database.

3) A module language and data manipulation language, for declaring the database procedures and executable statements of a specific database application program.

This standard defines the logical data structures and basic operations for an NDL database. It provides functional capabilities for designing, accessing, maintaining, controlling, and protecting the database.

This standard provides a vehicle for portability of database definitions and application programs between conforming implementations.

This standard specifies two levels. Level 2 is the complete NDL database language. Level 1 is the subset of Level 2 defined in clause 13, "Levels" on page 119.

This standard does not specify some of the facilities that might be provided in operational database environments, such as the following:

1) An access control facility for granting access and operational privileges to specific users

2) Additional integrity control capabilities for specifying more complex integrity constraints on the database

3) A facility to import and export schema definitions

4) A database unload facility for copying record and set populations to standard files for information interchange

5) A schema database for making schema and subschema information available to accessing application programs

6) A schema manipulation language for creating, modifying, or deleting portions of the schema or subschemas

7) Interfaces to a data dictionary

8) Application program pre-processing facilities for producing separate standard database modules and standard language programs

9) A data storage definition language for defining physical storage structures and physical access methods

10) Database procedures for user specified assertions and triggers

11) A natural language query facility for ad hoc access to the database

12) Report generator facilities for producing output tables and charts

13) Graphics capabilities for direct database interface with standard graphics systems

14) A distributed database facility for defining and accessing data at different nodes in a communications network

Standards for such additional facilities could be specified in a manner that is upward compatible with this standard. Some of the additional capabilities might be specified by subsequent versions of this standard, others could be specified by separate standards, and some may always be implementor-defined.

This standard applies to implementations that exist in an environment that may include application programming languages, end-user query languages, report generator systems, data dictionary systems, program library systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

2. References

ISO 1539, *Programming Languages - FORTRAN*, (endorsement by ISO of American National Standard X3.9).

ISO 1989, *Programming Languages - COBOL*, (endorsement by ISO of American National Standard X3.23).

ISO 6160, *Programming Languages - PL/I*, (endorsement by ISO of American National Standard X3.53).

ISO 7185, *Programming Languages - Pascal*, (endorsement by ISO of British Standard 6192 for the English text).

3. Overview

3.1 Organization

The organization of this standard is as follows:

1) 3.2, "Notation" on page 5 and 3.3, "Conventions" on page 6 define the notations and conventions used in this standard.

2) 3.4, "Conformance" on page 7 defines conformance criteria.

3) Clause 4, "Concepts" on page 9 defines terms and presents concepts used in the definition of NDL.

4) Clause 5, "Common elements" on page 17 defines language elements that occur in several parts of NDL language.

5) Clause 6, "Schema definition language" on page 31 defines the NDL facilities for specifying a database.

6) Clause 7, "Subschema definition language" on page 59 defines the NDL facilities for specifying portions of a database that can be referenced by an application program.

7) Clause 8, "Module language" on page 65 defines NDL modules and procedures.

8) Clause 9, "Data manipulation language" on page 75 defines the data manipulation statements of NDL.

9) Clause 10, "Auxiliary operations" on page 107 defines certain operations that are used in the definition of NDL data manipulation statements.

10) Clause 11, "Interpretive state" on page 111 defines the "session state", which is used in the definition of NDL data manipulation statements.

11) Clause 12, "Status codes" on page 117 defines the values of the STATUS parameter after the execution of NDL data manipulation statements.

12) Clause 13, "Levels" on page 119 defines the two levels of NDL.

3.2 Notation

The syntactic notation used in this standard is BNF ("Backus Normal Form" or "Backus-Naur Form"), with the following extensions:

1) Square brackets ([]) indicate optional elements.

2) Ellipses (...) indicate elements that may be repeated one or more times.

3) Braces ({}) group sequences of elements.

In the BNF syntax, a production symbol <A> is defined to "contain" a production symbol <B> if <B> occurs someplace in the expansion of <A>. If <A> contains <B>, then <B> is "contained in" <A>. If <A> contains <B>, then <A> is the "containing" <A> production symbol for <B>.

A production symbol <A> "immediately contains" a production symbol <B> if <A> contains <B> and if <B> occurs in the first expansion of <A>.

The lexical units of this standard (e.g. <separator>, <key word>, <identifier>, and <literal>) are defined elsewhere in this standard. The following characters have special significance in some production rules:

'	In <escape identifier>
"	In <character string literal>
(	In <comment>, <condition>, and <subscripts>
)	In <comment>, <condition>, and <subscripts>
*	In <comment>
	In <numeric literal>
-	In <numeric literal>
<	In <relation>
>	In <relation>
=	In <component identifier match>, <relation>, and <test database key equal statement>
E	In <approximate numeric literal>
_	In <identifier>
.	In <component identifier>, <component view identifier>, <numeric literal>, <subschema name clause>, and <subschema specification>

3.3 Conventions

Syntactic elements of this standard are specified in terms of:

1) *Function*: A short statement of the purpose of the element.

2) *Format*: A BNF definition of the syntax of the element.

3) *Syntax Rules*: Additional syntactic constraints not expressed in BNF that the element shall satisfy.

4) *General Rules*: A sequential specification of the run-time effect of the element.

In the Syntax Rules, the term "shall" defines conditions that are required to be true of syntactically conforming NDL language. The treatment of NDL language that does not conform to the Formats or the Syntax Rules is implementor-defined.

A conforming implementation is not required to perform the exact sequence of actions defined in the General Rules, but shall achieve the same effect on the database as that sequence.

The term "persistent object" is used to characterize objects such as <module>s and <schema>s that are created and destroyed by implementor-defined mechanisms.

In this standard, clauses begin a new odd-numbered page, and in clause 5, "Common elements" on page 17 through clause 11, "Interpretive state" on page 111 subclauses begin a new page. The resulting blank space is not significant.

3.4 Conformance

This standard specifies conforming NDL language and conforming NDL implementations. Conforming NDL language shall abide by the BNF Format and associated Syntax Rules. A conforming NDL implementation shall process standard conforming NDL language according to the General Rules.

A conforming implementation may provide additional facilities or options not specified by this standard. An implementation remains conforming even if it provides user options to process nonconforming NDL language or to process conforming NDL language in a nonconforming manner.

This standard does not define the method or the time of binding between application programs and database management system components.

An implementation that conforms to Level 1 of this standard shall meet the requirements for Level 2 conformance except that it may omit certain facilities as specified in clause 13, "Levels" on page 119.

This standard specifies three languages and defines the syntax and semantics of each of them. Implementors of database management systems may claim conformance for their products to this standard in two ways. An implementation may support the identical syntax of the languages and claim both syntactic and functional conformance, or it may use different syntax to express the same concepts, semantics, and functionality, and claim functional conformance only.

Implementations claiming functional conformance to any of the three standard languages shall specify equivalence with the semantics and functionality of those standard languages. For example, it should be possible to show that a schema displayed by a database management system display utility could be recoded using the standard schema definition language syntax and have equivalent functionality.

4. Concepts

4.1 Data type

A <data type> is a set of representable values. The logical representation of a value is a <literal>. The physical representation of a value is implementor-defined.

A value is primitive, in that it has no logical subdivision within this standard. Values are the basis of definition for data items and arrays.

A value is either a character string or a number. A character string and a number are not comparable values.

4.1.1 Character strings

A character string consists of a sequence of characters of the implementor-defined character set. A character string has a length, which is a positive integer that specifies the number of characters in the sequence.

4.1.2 Numbers

A number is either an exact numeric value or an approximate numeric value. All numbers are comparable values.

An exact numeric value has a precision and a scale. The precision is a positive integer that determines the number of significant decimal digits. The scale is a signed integer. A scale of 0 indicates that the number is an integer. For a scale of N, the exact numeric value is the integer value of the significant digits multiplied by 10 to the power -N.

An approximate numeric value consists of a mantissa and an exponent. The mantissa is a signed numeric value, and the exponent is a signed integer that specifies the magnitude of the mantissa. An approximate numeric value has a precision. The precision is a positive integer that specifies the number of significant binary digits in the mantissa.

Whenever an exact numeric value is assigned to a data item or parameter representing an exact numeric value, its value shall be exactly representable in the data type of the target. The value is converted to have the precision and scale of the target.

Whenever an exact or approximate numeric value is assigned to a data item or parameter representing an approximate numeric value, an approximation of its value is represented in the data type of the target. The value is converted to have the precision of the target.

4.2 Components

A component is either a data item or an array of data items. A <component type> defines a collection of component occurrences all having the same <data type>.

A data item consists of a single value. An array consists of a sequence of data items.

An array is specified by <extents>, which are a list of <unsigned integers>. The number of data items that occur in an array is equal to the product of the <unsigned integer>s.

A data item not contained in an array is referenced by a <component name> and a <database key>. A data item contained in an array is referenced by a <component name>, a <database key>, and <subscripts>.

All values of the same <component type> have the same <data type>. If the <component type> defines an array, then each component occurrence is an array with the same dimension and extent integers. All character strings of the same character string component have the same <length>. All numbers of the same exact numeric component have the same <precision> and <scale>. All numbers of the same approximate numeric component have the same <precision>.

Every <component type> is defined as part of a <record type>. The <component type> specifies the <data type>. If a <component type> defines an array, then it specifies the <extents>. A character string <component type> specifies the <length> of its character strings. An exact numeric <component type> specifies the <precision> and <scale> of its numbers. An approximate numeric <component type> specifies the <precision> of its numbers.

4.3 Records

The record is the basic unit of manipulation in this standard. Records are stored, erased, found, modified, and connected, disconnected, and reconnected in sets.

A record is a collection of components. A <record type> defines a collection of record occurrences all having the same <component type>s. A <record type> defines its <component type>s, and specifies integrity constraints that its record occurrences shall satisfy.

Each record is an occurrence of exactly one <record type>, and consists of exactly the components defined by that <record type>.

The records of each <record type> are maintained in an implementor-defined order.

4.3.1 Database keys

All records are distinguishable. Each record is identified uniquely by a <database key>. A <database key> is either *null*, in which case it identifies no record, or it is nonnull and identifies exactly one record in the database. A <database key> has no logical representation, and is not available through any interface defined by this standard. The physical representation of a <database key> is implementor-defined.

The <database key>s are used to model cursors in the session state and to define the semantics of statements in the data manipulation language. In the session state, a <database key> is referenced by <database key>. In data manipulation language statements, a <database key> is referenced by a <database key identifier>.

4.3.2 System record

This standard assumes the existence of a special SYSTEM <record type>. The SYSTEM <record type> has no <component type>s, no integrity constraints, and exactly one imaginary record occurrence, called the system record.

This standard does not specify a <database key> for the system record. The system record and SYSTEM <record type> exist only for the definition of singular sets.

4.4 Set types

A <set type> is a defined relationship between two or more <record type>s.

In a <set type>, one <record type> is designated as the *owner* <record type>, and each of the other <record type>s is referred to as a *member* <record type> of the <set type>.

For each <set type>, there may be zero, one, or more *sets* in a database.

A set consists of an *owner record*, whose <record type> is the owner <record type>, and zero, one, or more *member records*, each of whose <record type> is a member <record type>.

A set is identified uniquely in a database by its <set name> and its owner record occurrence. When an occurrence of the owner <record type> of a <set type> is stored in the database, a new set of that <set type> is created.

When an occurrence of <record type> is stored in the database, it belongs to at most one set of each <set type> in which the <record type> is a member <record type>.

The member records of each set of a <set type> are maintained in a sequential order determined by the ordering criteria for that <set type>.

4.4.1 Singular sets

A singular <set type> is a <set type> having SYSTEM as its owner <record type>. A singular set is the unique occurrence of a singular <set type>.

4.4.2 Recursive sets

A recursive <set type> is a set type having the same <record type> as both the owner <record type> and as a member <record type>. A recursive set is an occurrence of a recursive <set type>.

4.5 Schema

A <schema> is a persistent object specified by the schema definition language. The <schema> is a logical description of the database. A <schema> consists of a <schema name>, a collection of <record type> descriptions, and a collection of <set type> descriptions.

4.6 Subschemas

A <subschema> is a persistent object specified by the subschema definition language. The <subschema> is a logical description of that portion of a database available to an accessing <module>. A <subschema> consists of a <subschema name>, a collection of <record view>s, and a collection of <set view>s.

A <record view> specifies the available <component type>s of a given <record type>. A <set view> specifies an available <set type>. The <record view>s and <set view>s taken together determine the available member <record type>s of each <set type>.

The <record view>s and <set view>s may define alias <component view name>s, <record view name>s and <set view name>s for <component type>s, <record type>s, and <set type>s. The alias names become the only names available for those <component type>s, <record type>s, and <set type>s to an accessing <module>.

The records available to an accessing <module> are maintained in an implementor-defined order. This order is not necessarily the same as the implementor-defined order of the records of each <record type>.

4.7 Database

A database is the collection of all data defined by a <schema>. It consists of the occurrences of each <record type> and each <set type>. There is exactly one database for each <schema> definition.

4.8 Modules

A <module> is a persistent object specified in the module language. A <module> consists of an optional <module name>, a <language clause>, a <subschema specification>, an optional <temporary set specifications>, and one or more <procedure>s.

An application program is a segment of executable code, possibly consisting of multiple sub-programs. A single <module> is associated with an application program during its execution. An application program shall be associated with at most one <module>. The manner in which this association is specified, including the possible requirement for execution of some implementor-defined statement, is implementor-defined.

4.9 Procedures

A <procedure> consists of a <procedure name>, an optional sequence of <parameter declaration>s, and an optional sequence of <NDL statement>s.

An application program associated with a <module> may reference the <procedure>s of that <module> by a "call" statement that specifies the <procedure name> of the <procedure> and supplies a sequence of parameter values corresponding in number and in <data type> to the <parameter declaration>s of the procedure. A call of a <procedure> causes the sequence of <NDL statement>s that it contains to be executed.

The <subschema specification> of a <module> specifies the <subschema> that defines the records and sets that can be referenced by the <module>.

The <temporary set specifications> of a <module> defines temporary <set type>s that may be referenced by the <module>.

4.10 Standard programming languages

This standard specifies the actions of <procedure>s in <module>s when those <procedure>s are called by programs that conform to specified standard programming languages. The terms "standard COBOL program", "standard FORTRAN program", "standard Pascal program", and "standard PL/I program" refer to programs that meet the conformance criteria of the standards listed in clause 2, "References" on page 3.

4.11 Temporary sets

A <temporary set> is an occurrence of a temporary <set type>. A temporary <set type> is declared by a <temporary set specification> in a <module>.

For each <temporary set specification> in a <module>, an occurrence of the corresponding temporary <set type> is created when the <module> is initiated and destroyed when the <module> is terminated.

4.12 Parameters

A parameter is declared in a <procedure> by a <parameter declaration>. The <parameter declaration> specifies whether the parameter is an elementary data item or an array of data items, and specifies the <data type> of its values. A parameter either assumes or supplies the value or values of the corresponding argument in the call of that procedure.

4.12.1 STATUS parameter

The STATUS parameter is a special parameter of <length> 5 characters. If such a parameter is declared in a <procedure>, then its value is set to a status code that either indicates that a call of the <procedure> was successfully completed or that identifies an exception condition that occurred during execution of the <procedure>.

4.12.2 TEST parameter

The TEST parameter is a special parameter of <length> 1 character. If a test statement is executed in a <procedure> containing this parameter, then the value of the parameter is set to "1" if the test is true, and to "0" if the test is false.

4.12.3 RECORD parameter

The RECORD parameter is a special parameter of <length> 18 characters. If such a parameter is declared in a <procedure>, then it is set to the <record name> of the record identified by the <session cursor> of the <session state>.

4.13 Session

A database operation is the execution of an <NDL statement>.

A session is the sequence of database operations performed during the execution of an application program associated with a <module>.

A <session state> is an ephemeral object associated with a session. A <session state> is created prior to the first database operation in a session and is destroyed after the last database operation in the session.

A <session state> consists of <cursors>, <temporary sets>, and a <ready list>. The <cursors> include: A <session cursor> to identify the current session record, a <record cursor> to identify the current record of each <subschema> <record type>, and a <set cursor> to identify the owner record and the current member record in the current set of each <subschema> <set type>. The <temporary sets> maintain the contents of each <temporary set> defined by the associated <module>. The <ready list> maintains the <ready specification>s for each <record type> that has been activated by a <ready statement>.

The <session state> is available to an application program only through <NDL statement>s. The contents of the <session state> are modified by the database management system upon execution of <NDL statement>s in a <procedure> of its associated <module>. The data manipulation language includes <NDL statement>s to find and compare <database key>s and to modify the contents of the <ready list> and each <temporary set>.

4.14 Transactions

A transaction is a sequence of operations, including database operations, that is atomic with respect to recovery and concurrency. Transactions terminate with a <commit statement> or a <rollback statement>. If a transaction terminates with a <commit statement>, then all changes made to the database by that transaction are made accessible to all concurrent transactions. If a transaction terminates with a <rollback statement>, then all changes made to the database by that transaction are canceled. Committed changes cannot be canceled. Changes made to the database by a transaction can be perceived by that transaction, but cannot be perceived by other transactions until that transaction terminates with a <commit statement>.

The execution of concurrent transactions is guaranteed to be serializable. A serializable execution is defined to be an execution of the operations of concurrently executing transactions that produces the same effect as some serial execution of those same transactions. A serial execution is one in which each transaction executes to completion before the next transaction begins.

The execution of an <NDL statement> within a transaction has no effect on the database or <session state> other than the effect stated in the General Rules for that <NDL statement>. Together with serializable execution, this implies that all read operations are reproducible within a transaction, except for changes explicitly made by the transaction itself.

4.15 Integrity constraints

Integrity constraints restrict the valid states of the database. Integrity constraints may be defined for <record type>s or <set type>s, and for component constraints between owner and member <record type>s.

Integrity constraints are checked after execution of each <NDL statement>. If the objects associated with the integrity constraint do not satisfy the constraint, then the <NDL statement> has no effect on the database, and the STATUS parameter is set to indicate the specified exception.

4.15.1 Check conditions

A check condition is an expression that shall be satisfied by the values of a record when its status in the database is altered. The expression is specified by a <condition> of a <record check clause> of a <record type> or a <member check clause> of a <set type>.

4.15.2 Default values

A default value is a value to be assumed by all component occurrences in the absence of an explicit value supplied by a <procedure>. A default value is specified by a <literal> in a <default clause> of a <component type>.

4.15.3 Uniqueness constraints

A uniqueness constraint is a specification that no two records of a given <record type>, or no two member records of a set of a given <set type>, may occur in the database with identical values for specified components. Uniqueness constraints are specified by <record uniqueness clause>s in a <record type>, or by <member uniqueness clause>s in a <member clause> of a <set type>.

4.15.4 Set ordering criteria

A set ordering specifies the sequential ordering of member records in a set. The ordering is one of the following:

1) Sorted by component values

2) Chronological or reversed chronological

3) Relative positioning as next or prior to a given member record

4) Implementor-defined

Set ordering criteria are specified by the <order clause> of a <set type>, and if the order is sorted, by the <key clause> of each <member clause> contained in that <set type>.

4.15.5 Set membership

Set membership specifies the insertion and retention modes of a record for each set in which it occurs as a member record. Each member <record type> has an insertion mode and a retention mode.

Insertion is either automatic, structural, or manual. If insertion is manual, then records shall be inserted as member records in a set only by an explicit <connect statement>. If insertion is automatic or structural, then each record of a member <record type> becomes a member of some set of that <set type> when it is initially stored in the database. If insertion is automatic or manual, then the owner record is identified by the application program. If insertion is structural, then the owner record is selected by the database management system to have values of specified data items equal to those of the record to be inserted. The insertion mode is specified by an <insertion clause> in each <member clause> of the <schema>.

Retention is either fixed, mandatory, or optional. If retention is fixed, then a record, having once become a member record of some set, remains a member of that set until it is erased from the database. If retention is mandatory, then a record, having once become a member record of a set, remains a member of some set of the same <set type> until it is erased from the database. If retention is optional, then a record, having once

become a member record of some set, need not remain a member of that set or of any set of that <set type>. The retention mode is specified by a <retention clause> in each <member clause> of the <schema>.

5. Common elements

5.1 <condition>

Function

Specify an expression that shall be evaluated as either *true* or *false*.

Format

<condition> ::=
<alternative> [{OR <alternative>}...]

<alternative> ::=
<simple condition> [{AND <simple condition>}...]

<simple condition> ::=
<subcondition>
| <negated subcondition>
| <relation condition>

<subcondition> ::= (<condition>)

<negated subcondition> ::=
NOT (<condition>)

<relation condition> ::=
<operand> <relation> <operand>

<relation> ::=
< | <= | = | >= | > | <>

Syntax Rules

1) If the first <operand> in a <relation condition> specifies an array, then the second <operand> in that <relation condition> shall not specify an array.

2) If the type of the first operand is character string, then the type of the second operand shall be character string. If the type of the first operand is exact numeric, then the type of the second operand shall be exact numeric or approximate numeric. If the type of the first operand is approximate numeric, then the type of the second operand shall be exact numeric or approximate numeric.

General Rules

1) A <condition> is *true* if any <alternative> contained in it is *true*; otherwise, it is *false*.

2) An <alternative> is *true* if every <simple condition> contained in it is *true*; otherwise, it is *false*.

3) A <simple condition> that immediately contains a <subcondition> is *true* if the <subcondition> is *true*; otherwise, it is *false*.

A <simple condition> that immediately contains a <negated subcondition> is *true* if the <condition> immediately contained in the <negated subcondition> is *false*; otherwise, it is *false*.

A <simple condition> that immediately contains a <relation condition> is *true* if the <relation condition> is *true*; otherwise, it is *false*.

4) The first (second) operand of a <relation condition> is the value of the first (second) <operand> in that <relation condition>.

5) The <relation> of a <relation condition> specifies a comparison to be made between the two operands of the <relation condition>. The comparisons specified by the alternative <relation>s are as follows:

<relation>	Comparison
<	less than
<=	less than or equal
=	equal
<>	not equal
>	greater than
>=	greater than or equal

6) If neither operand of a <relation condition> is an array, then the <relation condition> is *true* if the specified comparison of the two operands is *true*; otherwise, the <relation condition> is *false*. If one of the operands of a <relation condition> is an array, then the <relation condition> is *true* if the specified comparison of every value in the array sequence with the other operand is *true*; otherwise, the <relation condition> is *false*.

7) Case:

   a) If operands are of exact numeric or approximate numeric type, then the comparison is performed according to their numeric values.

   b) If the operands are of character string type, then:

      i) If the lengths of the two operands are not equal, then the shorter operand is considered to be extended on the right with space characters.

      ii) The comparison is performed from left to right, comparing the individual characters of the operands. If all such individual comparisons are equal, then the two operands are equal; otherwise, the result of the comparison of the operands is defined to be the result of the leftmost nonequal comparison of the individual characters.

      iii) The comparison of individual characters is implementor-defined.

8) In General Rules, the terms "less than", "greater than", "equal", and "not equal" refer to the corresponding relational operators.

5.2 <operand>

Function

Specify a value.

Format

<operand> ::=
<component identifier>
| <component view identifier>
| <parameter identifier>
| <literal>

Syntax Rules

1) Case:

   a) If an <operand> is contained in a <schema>, then it shall not be a <parameter identifier> or a <component view identifier>.

   b) If an <operand> is contained in a <module>, then it shall not be a <component identifier>.

2) The data type of an <operand> is the data type of the <component identifier>, <component view identifier>, <parameter identifier>, or <literal>.

General Rules

1) The value of an <operand> is the data item or array referenced by the <component identifier>, <component view identifier>, or <parameter identifier>, or the value of the <literal>.

5.3 <identifier>

Function

Specify names for parameters in a <procedure> or objects in the database.

Format

<identifier> ::=
<regular identifier> | <escape identifier>

<regular identifier> ::=
<upper case letter>
[{[<underscore>] <letter or digit>}...]

<underscore> ::= _

<letter or digit> ::=
<upper case letter> | <digit>

<letter> ::=
<upper case letter> | <lower case letter>

<upper case letter> ::=
A | B | C | D | E | F | G | H | I
| J | K | L | M | N | O | P | Q | R
| S | T | U | V | W | X | Y | Z

<lower case letter> ::=
a | b | c | d | e | f | g | h | i
| j | k | l | m | n | o | p | q | r
| s | t | u | v | w | x | y | z

<digit> ::=
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<escape identifier> ::=
'<escape identifier character representation>...'

<escape identifier character representation> ::=
<escape identifier character>
| <apostrophe representation>

<escape identifier character> ::=
See Syntax Rule 3.

<apostrophe representation> ::= ''

Syntax Rules

1) A <regular identifier> shall contain at most 18 characters. An <escape identifier> shall contain at most 18 <escape identifier character representation>s.

2) A <regular identifier> shall not be the same sequence of characters as a <key word> (see 5.8, "Comments, spaces, and key words" on page 28).

3) An <escape identifier character> is any character in the implementor-defined character set other than the apostrophe character (').

4) Each <apostrophe representation> in an <escape identifier> represents a single apostrophe character.

5) If an <escape identifier> (apart from the enclosing apostrophes) conforms to the format of a <regular identifier>, and if it is not a <key word>, then it is equivalent to the <regular identifier> formed by removing the enclosing apostrophes.

General Rules

None.

5.4 <literal>

Function

Specify a literal value.

Format

<literal> ::=
<character string literal>
| <numeric literal>

<character string literal> ::=
"<character representation>..."

<character representation> ::=
<nonquote character>
| <quote representation>

<nonquote character> ::=
See Syntax Rule 1.

<quote representation> ::=
""

<numeric literal> ::=
<exact numeric literal>
| <approximate numeric literal>

<exact numeric literal> ::=
[+ | -] { <unsigned integer>[.<unsigned integer>]
| <unsigned integer>.
| .<unsigned integer>}

<approximate numeric literal> ::=
<mantissa>E<exponent>

<mantissa> ::= <exact numeric literal>

<exponent> ::= <signed integer>

<signed integer> ::= [+ | -] <unsigned integer>

<unsigned integer> ::=
<digit>...

Syntax Rules

1) A <nonquote character> is any character in the implementor-defined character set other than the double quote mark character (").

2) The data type of a <character string literal> is character string. The length of a <character string literal> is the number of <character representation>s that it contains. Each <quote representation> in a <character string literal> represents a single quotation mark character in both the value and the length of the <character string literal>.

3) An <exact numeric literal> without a decimal point (.) has an implied decimal point following the last <digit>.

4) The data type of an <exact numeric literal> is exact numeric. The precision of an <exact numeric literal> is the number of <digit>s that it contains. The scale of an <exact numeric literal> is the number of <digit>s to the right of the decimal point.

5) The data type of an <approximate numeric literal> is approximate numeric. The precision of an <approximate numeric literal> is the number of binary digits required to represent the significant digits of its mantissa.

General Rules

1) The value of a <character string literal> is the sequence of characters that it contains.

2) The numeric value of an <unsigned integer>, <signed integer>, or <exact numeric literal> is derived from the normal mathematical interpretation of signed positional decimal notation.

3) The numeric value of an <approximate numeric literal> is the product of the exact numeric value represented by the <mantissa> with the number obtained by raising the number 10 to the power represented by the <exponent>.

5.5 <data type>

Function

Specify a data type.

Format

<data type> ::=
<character string type>
| <exact numeric type>
| <approximate numeric type>

<character string type> ::=
CHARACTER [<length>]

<exact numeric type> ::=
FIXED <precision> [<scale>]
| NUMERIC <precision> [<scale>]
| INTEGER

<approximate numeric type> ::=
FLOAT <precision>
| REAL
| DOUBLE PRECISION

<length> ::= <unsigned integer>

<precision> ::= <unsigned integer>

<scale> ::= <signed integer>

Syntax Rules

1) The value of an <unsigned integer> that is a <length> or a <precision> shall be greater than 0.

2) If a <length> is omitted, then it is assumed to be 1. If a <scale> is omitted, then it is assumed to be 0.

General Rules

1) A <data type> specifies a class of data item values.

2) A data item of <data type> CHARACTER has a character string value with length equal to the value of the specified <length>.

3) A data item of <data type> FIXED has an exact numeric value with scale equal to the value of the specified <scale>, and with precision greater than or equal to the value of the specified <precision>.

4) A data item of <data type> NUMERIC has an exact numeric value with precision and scale equal respectively to the value of the specified <precision> and <scale>. If the precision is P and the scale is S, then the absolute value of any nonzero NUMERIC data item shall be greater than or equal to 1E-S and less than or equal to 1E(P-S)-1E-S.

5) A data item of <data type> INTEGER has an exact numeric value with a scale of 0 and with implementor-defined precision.

6) A data item of <data type> FLOAT has an approximate numeric value with binary precision equal to or greater than the value of the specified <precision>.

7) A data item of <data type> REAL has an approximate numeric value with implementor-defined precision.

8) A data item of <data type> DOUBLE PRECISION has an approximate numeric value with implementor-defined precision that is greater than the implementor-defined precision of REAL.

5.6 <occurs clause>

Function

Define an array by specifying its <extents>.

Format

<occurs clause> ::=
OCCURS <extents>

<extents> ::= <unsigned integer>...

Syntax Rules

1) The value of each <unsigned integer> in the <extents> shall be greater than 0.

General Rules

1) An <occurs clause> specifies an array of data items. The product of the values of the <unsigned integer>s in the <extents> specifies the number of data items contained in each occurrence of the array.

5.7 <subscripts>

Function

Specify a data item of an array.

Format

<subscripts> ::=
(<operand>...)

Syntax Rules

1) The *contextual* <occurs clause> of a <subscripts> is specified in the Syntax Rules of the production symbol that immediately contains the <subscripts>.

2) The number of <operand>s shall be equal to the number of <unsigned integer>s in the <extents> of the contextual <occurs clause>.

3) Case:

   a) If an <operand> is a <literal>, then it shall be an <unsigned integer> whose value is positive and is not greater than the value of the corresponding <unsigned integer> of the <extents> of the contextual <occurs clause>.

   b) If an <operand> is not a <literal>, then its data type shall be exact numeric with a scale of 0.

General Rules

1) Let d be the number of <operand>s in the <subscripts>.

2) For i from 1 to d, let Si be the value of the i-th <operand> in the <subscripts>.

3) For i from 1 to d, let Ei be the value of the i-th <unsigned integer> in the <extents> of the contextual <occurs clause>.

4) For i from 1 to d, if Si is less than 1 or Si is greater than Ei, then raise exception *array reference: subscript out of bounds*.

5) Let Md be 1. For i from 1 to (d-1), let Mi be the product of the terms Ej, for j from (i+1) to d.

6) Let J be the sum of the terms Mi*(Si-1), for i from 1 to d.

7) The value of the <subscripts> is J+1.

5.8 Comments, spaces, and key words

Function

Specify lexical units.

Format

<separator> ::=
{ <comment> | <space> | <newline> }...

<comment> ::=
(*[<character>...]*)

<character> ::=
<digit> | <letter> | <special character>

<special character> ::=
See Syntax Rule 1.

<space> ::=
space character

<newline> ::=
implementor-defined end-of-line indicator

<word> ::=
<key word> | <identifier> | <literal>

<key word> ::=
ABSOLUTE | ALL | AND | AS | ASCENDING | AUTOMATIC
| CASCADE | CHARACTER | CHECK | COBOL
| COMMIT | CONNECT | CONTAINS
| DEFAULT | DESCENDING | DISCONNECT | DOUBLE | DUPLICATES
| EMPTY | ERASE | EXCLUSIVE
| FIND | FINISH | FIRST | FIXED | FLOAT | FOR | FORTRAN | FROM | FULL
| GET | IN | INSERTION | INTEGER | ITEM | KEY | LANGUAGE | LAST
| MANDATORY | MANUAL | MEMBER | MODIFY | MODULE
| NEXT | NOT | NULL | NULLIFY | NUMERIC
| OCCURS | OF | OPTIONAL | OR | ORDER | OWNER
| PARTIAL | PASCAL | PLI | PRECISION
| PRIOR | PROCEDURE | PROHIBITED | PROTECTED
| READY | REAL | RECONNECT | RECORD | RELATIVE | RENAMED
| RETAIN | RETENTION | RETRIEVE | ROLLBACK
| SCHEMA | SESSION | SET | SHARED | SORTED
| STATUS | STORE | STRUCTURAL | SUBSCHEMA | SYSTEM
| TEST | TO | TYPE | UNIQUE | UPDATE | WHERE | WITH

Syntax Rules

1) A <special character> is any character in the implementor-defined character set other than a <digit> or a <letter>. If the implementor-defined end-of-line indicator (<newline>) is a character, then it is also excluded from <special character>.

2) A terminal production of <schema>, <subschema>, or <module> consists of a sequence of <word>s and symbols optionally separated by <separator>s. A <word> shall not be immediately followed by another <word> without an intervening <separator>.

3) The string "<character>..." within a <comment> shall not contain the substring "*)".

General Rules

None.

6. Schema definition language

6.1 <schema>

Function

Define the logical structure of a database.

Format

<schema> ::=
<schema name clause>
[{<record type> | <set type>}...]

Syntax Rules

None.

General Rules

None.

6.2 <schema name clause>

Function

Name a <schema>.

Format

<schema name clause> ::=
SCHEMA <schema name>

<schema name> ::= <identifier>

Syntax Rules

1) The <schema name> shall be different from the <schema name> of any other <schema> in the same environment. The concept of environment is implementor-defined.

General Rules

1) A <schema name clause> defines the <identifier> to be a <schema name> that designates the containing <schema>.

6.3 <record type>

Function

Define a <record type>.

Format

<record type> ::=
<record name clause>
[{ <record uniqueness clause>
| <component type>
| <record check clause>}...]

Syntax Rules

None.

General Rules

None.

6.4 <record name clause>

Function

Name a <record type>.

Format

<record name clause> ::=
RECORD <record name>

<record name> ::= <identifier>

Syntax Rules

1) The <record name> shall be different from the <record name> of any other <record name clause> in the containing <schema>.

General Rules

1) A <record name clause> defines the <identifier> to be a <record name> that designates the containing <record type>.

6.5 <record uniqueness clause>

Function

Specify a uniqueness constraint for occurrences of a <record type>.

Format

<record uniqueness clause> ::=
UNIQUE <component identifier>...

Syntax Rules

1) If the <component name> of a <component identifier> designates an array, then that <component identifier> shall contain <subscripts>.

2) A given <component identifier> shall be contained at most once in a single <record uniqueness clause>.

3) The contextual <record type> of each <component identifier> in a <record uniqueness clause> is the containing <record type>.

General Rules

1) When the <component identifier>s of a <record uniqueness clause> are referenced for a record occurrence, the contextual <database key> for those <component identifier>s is the <database key> of that record occurrence.

2) A <record uniqueness clause> is violated if the database contains two occurrences of the containing <record type> in which the values of each of the components designated by the <component identifier>s in the <record uniqueness clause> in the first record occurrence are *equal* to the values of the corresponding components in the second record occurrence.

6.6 <component type>

Function

Define a <component type>.

Format

<component type> ::=
<component name clause>
<data type>
[<occurs clause>]
[<default clause>]

Syntax Rules

None.

General Rules

None.

6.7 <component name clause>

Function

Name a <component type>.

Format

<component name clause> ::= ITEM <component name>

<component name> ::= <identifier>

Syntax Rules

1) The <component name> shall be different from the <component name> of any other <component name clause> in the containing <record type>.

General Rules

1) A <component name clause> defines the <identifier> to be a <component name> that designates the containing <component type>.

6.8 <default clause>

Function

Specify a default value for a <component type>.

Format

<default clause> ::=
DEFAULT <literal>

Syntax Rules

1) The subject <data type> of a <default clause> is the <data type> of the containing <component type>.

2) Case:

   a) If the subject <data type> defines character string values, then the <literal> shall be a <character string literal>. The length of the <character string literal> shall not be greater than the value of the <length> of the subject <data type>.

   b) If the subject <data type> defines exact numeric values, then the <literal> shall be an <exact numeric literal>. The number of significant digits to the right of the decimal point shall not be greater than the <scale> of the subject <data type>. The number of significant digits to the left of the decimal point shall not be greater than P-S, where P and S are respectively the <precision> and <scale> of the subject <data type>.

   c) If the subject <data type> defines approximate numeric values, then the <literal> shall be an <approximate numeric literal> or an <exact numeric literal>. The number of binary digits required to represent the significant digits shall not be greater than the <precision> of the subject <data type>.

General Rules

1) When an occurrence of the containing <record type> is stored by a <store statement>, the value of each data item contained in the occurrence of the containing <component type> is as follows:

   Case:

   a) If the <literal> is an <exact numeric literal>, then the value is the exact numeric value of the <literal>.

   b) If the <literal> is an <approximate numeric literal>, then the value is the approximate numeric value of the <literal>.

   c) If the <literal> is a <character string literal> and the length of the <literal> is equal to the length of the subject <data type>, then the value is the <literal>.

   d) If the <literal> is a <character string literal> and the length of the <literal> is less than the length of the subject <data type>, then the value is the <character string literal>, extended on the right to the length of the subject component with space characters.

6.9 <record check clause>

Function

Specify a validity condition for occurrences of a <record type>.

Format

<record check clause> ::=
CHECK <condition>

Syntax Rules

1) The contextual <record type> of each <component identifier> in the <condition> is the containing <record type>.

General Rules

1) When the <condition> of a <record check clause> is evaluated for a record occurrence, the contextual <database key> for any <component identifier> in the <condition> is the <database key> of that record occurrence.

2) A <record check clause> is violated if the database contains a record occurrence of the containing <record type> for which the <condition> is *false*.

6.10 <set type>

Function

Define a <set type>.

Format

<set type> ::=
<set name clause>
<owner clause>
<order clause>
<member clause>...

Syntax Rules

None.

General Rules

None.

6.11 <set name clause>

Function

Name a <set type>.

Format

<set name clause> ::=
SET <set name>

<set name> ::= <identifier>

Syntax Rules

1) The <set name> shall be different from the <set name> of any other <set name clause> in the containing <schema>.

General Rules

1) A <set name clause> defines the <identifier> to be a <set name> that designates the containing <set type>.

6.12 <owner clause>

Function

Specify the owner of a <set type>.

Format

<owner clause> ::=
OWNER {<record name> | SYSTEM}

Syntax Rules

1) If an <owner clause> contains a <record name>, then that <record name> shall designate a <record type> in the containing <schema>.

General Rules

1) Case:

   a) If the <owner clause> specifies a <record name>, then that <record name> is the owner <record name> of the containing <set type>, and the <record type> designated by that <record name> is the owner <record type> of that <set type>.

   b) If the <owner clause> specifies SYSTEM, then the containing <set type> is a singular <set type> with SYSTEM as its owner <record type>.

6.13 <order clause>

Function

Specify the ordering of member records in a set.

Format

```
<order clause> ::=
        ORDER <order option>

<order option> ::=
          FIRST
        | LAST
        | NEXT
        | PRIOR
        | DEFAULT
        | <sorted order>

<sorted order> ::=
        SORTED {<record type sequence> | <order duplicates>}

<record type sequence> ::=
        RECORD TYPE <record name>...

<order duplicates> ::=
        DUPLICATES
            {  PROHIBITED
             | FIRST
             | LAST
             | DEFAULT }
```

Syntax Rules

1) If <sorted order> is specified, then a <key clause> shall be included in each <member clause> of the containing <set type>. Otherwise, no <member clause> of the containing <set type> shall include a <key clause>.

2) If <record type sequence> is specified, then each <record name> shall designate a member <record type> of the containing <set type>, and every <record name> that designates a member <record type> of the containing <set type> shall occur exactly once in the <record type sequence>.

3) If <record type sequence> is included, then the <key clause> of each <member clause> in the containing <set type> shall include a <record type key item>; otherwise, no such <key clause> shall include a <record type key item>.

4) If <order duplicates> is specified, then no <key clause> of any <member clause> in the containing <set type> shall include a <key duplicates>; otherwise, each <key clause> of every <member clause> in the containing <set type> shall include a <key duplicates>.

NOTE: *Together with rules governing the <key clause>, this ensures that set members with duplicate key items are controlled either by a single <order duplicates> in the <order clause> or by a <key duplicates> in each <member clause>.*

General Rules

1) If the <order option> is LAST or FIRST, then the database control system shall maintain member records of an occurrence of the containing <set type> respectively in chronological or reverse chronological order. This chronology is based on the most recent time at which an <insert operation> inserts a member record into the set occurrence. An <insert operation> resulting from a <modify statement> or <reconnect statement> that (re)inserts a member record into the same set occurrence establishes a new point in the chronology for LAST and FIRST.

2) If the <order option> is NEXT or PRIOR, then the database control system maintains the member records of an occurrence of the containing <set type> in an order determined by the <position> contained in the object <set cursor> of the <insert operation>.

3) If the <order option> is DEFAULT, then the database control system shall maintain the member records of an occurrence of the containing <set type> in an implementor-defined order. This order is subject to the reproducibility requirement within a transaction, but it may change between transactions.

4) If <sorted order> is specified, then the database control system shall maintain the member records of an occurrence of the containing <set type> in an order based on the sort control key of each member record, where the sort control key of a member record consists of a sequence of key items as specified in the <key clause> of the corresponding <member clause> of the containing <set type>.

5) If <record type sequence> is specified, then the member <record type>s of the containing <set type> shall be used as key items in the sort control keys defined in the <key clause>s of the <member clause>s of the containing <set type>. The sequence of <record name>s in the <record type sequence> specifies the ascending order of <record type>s; the descending order of <record type>s is given by the reverse of that sequence.

6) An <order duplicates> specifies the action taken by the database control system when an <insert operation> would cause an occurrence of the containing <set type> to contain two or more member records that have *equal* values for all of the key items:

   Case:

   a) If PROHIBITED is specified, then the <order clause> is violated.

   b) If LAST or FIRST is specified, then member records that have *equal* values for the key items shall be ordered respectively in chronological or reverse chronological sequence. This chronology is based on the most recent time that a <store statement>, <modify statement>, <connect statement>, or <reconnect statement> inserted a given member record into the set occurrence, or a <modify statement> specified values for one or more key items in the given member record. A <modify statement> that specifies values for one or more key items in a member record that are *equal* to the current values for those key items in that record establishes a new point in the chronology for <order duplicates> FIRST and LAST.

   c) If DEFAULT is specified, then member records that have *equal* values for all of the key items shall be ordered in an implementor-defined sequence.

   NOTE: *If <order duplicates> is not specified, then the action to be taken is controlled by the <key duplicates> of each <member clause> of the containing <set type>.*

6.14 <member clause>

Function

Specify a member <record type> of a <set type>.

Format

<member clause> ::=
<member record name clause>
<insertion clause>
<retention clause>
[<member uniqueness clause>...]
[<key clause>]
[<member check clause>...]

Syntax Rules

None.

General Rules

None.

6.15 <member record name clause>

Function

Specify the <record name> of a member <record type> of a <set type>.

Format

<member record name clause> ::=
MEMBER <record name>

Syntax Rules

1) The <record name> shall designate a <record type> in the containing <schema>.

2) The <record name> shall not designate any other member <record type> in the containing <set type>.

General Rules

1) A <member record name clause> defines the <record type> designated by the <record name> to be a member <record type> of the containing <set type> and to be the member <record type> of the containing <member clause>.

6.16 <insertion clause>

Function

Define the insertion characteristics of a member <record type> of a <set type>.

Format

<insertion clause> ::=
INSERTION <insertion mode>

<insertion mode> ::=
AUTOMATIC
| STRUCTURAL <structural specification>
| MANUAL

<structural specification> ::=
<component identifier match>
[{AND <component identifier match>}...]

<component identifier match> ::=
<member component identifier> = <owner component identifier>
| <owner component identifier> = <member component identifier>

<member component identifier> ::= <component identifier>

<owner component identifier> ::= <component identifier>

Syntax Rules

1) If the <insertion clause> specifies STRUCTURAL, then:

   a) If the <component name> of a <component identifier> designates an array, then that <component identifier> shall contain <subscripts>.

   b) The containing <set type> shall not be a singular <set type>. The object <record type> is the owner <record type> of the containing <set type>. The subject <record type> is the member <record type> of the containing <member clause>.

   c) If the subject <record type> and the object <record type> are the same, then every <component identifier> in the <component identifier match> shall contain a <qualifier> that specifies either OWNER or MEMBER.

   d) Any <component identifier> in a <component identifier match> that contains a <component name> that is defined in both the subject <record type> and the object <record type> shall contain a <qualifier>.

   e) If a <component identifier> in a <component identifier match> contains a <qualifier> that specifies a <record name>, then that <record name> shall be the <record name> of either the subject <record type> or the object <record type>; the contextual <record type> of the <component identifier> is that <record type>.

f) If a <component identifier> in a <component identifier match> contains a <qualifier> that specifies OWNER (MEMBER), then the contextual <record type> of the <component identifier> is the object (subject) <record type>.

g) If a <component identifier> in a <component identifier match> does not contain a <qualifier>, then the <component name> of that <component identifier> shall be defined in either the subject <record type> or the object <record type>; the contextual <record type> of the <component identifier> is that <record type>.

h) The data items referenced by <owner component identifier>s shall be exactly the <component identifier>s listed in a <record uniqueness clause> in the object <record type>.

i) The data items referenced by <member component identifier>s shall be data items of the subject <record type>. The <data type> of the <component type> referenced by the <member component identifier> of a <component identifier match> shall be identical to the <data type> of the <component type> referenced by the <owner component identifier> of that <component identifier match>.

General Rules

1) If STRUCTURAL is specified, then let M1, M2,... be the <component identifier>s specified as <member component identifier>s. The contextual <database key> of those <component identifier>s is the <database key> of the new member record occurrence. Let O1, O2,... be the <component identifier>s specified as <owner component identifier>s. The contextual <database key> of those <component identifier>s is the <database key> of the owner record occurrence of the set occurrence specified by the <insert operation>. An implicit <member check clause> of the following form is enforced:

CHECK M1 = O1 AND M2 = O2 ...

6.17 <retention clause>

Function

Define the retention characteristics of a member <record type> of a <set type>.

Format

<retention clause> ::=
RETENTION {FIXED | MANDATORY | OPTIONAL}

Syntax Rules

None.

General Rules

None.

6.18 <member uniqueness clause>

Function

Specify a uniqueness constraint for member records of each occurrence of a <set type>.

Format

<member uniqueness clause> ::=
UNIQUE <component identifier>...

Syntax Rules

1) If the <component name> of a <component identifier> designates an array, then that <component identifier> shall contain <subscripts>.

2) A given <component identifier> shall be contained at most once in a single <member uniqueness clause>.

3) The subject <record type> of a <member uniqueness clause> is the member <record type> of the containing <member clause>.

   The contextual <record type> of each <component identifier> in a <member uniqueness clause> is the subject <record type>.

General Rules

1) When the <component identifier>s of a <member uniqueness clause> are referenced for a record occurrence, the contextual <database key> for those <component identifier>s is the <database key> of that record occurrence.

2) A <member uniqueness clause> is violated if the database contains an occurrence of the containing <set type> that contains two member occurrences of the subject <record type> in which the values of each of the components designated by the <component identifier>s in the <member uniqueness clause> in the first record occurrence are *equal* to the values of the corresponding components in the second record occurrence.

6.19 <key clause>

Function

Define the sort control key for a member <record type> of a sorted <set type>.

Format

<key clause> ::=
KEY <key item>... [<key duplicates>]

<key item> ::=
{ASCENDING | DESCENDING}
{<component identifier> | <record type key item>}...

<record type key item> ::= RECORD TYPE

<key duplicates> ::=
DUPLICATES
{ PROHIBITED
| FIRST
| LAST
| DEFAULT }

Syntax Rules

1) If the <component name> of a <component identifier> designates an array, then that <component identifier> shall contain <subscripts>.

2) A given <component identifier> shall be contained at most once in a single <key clause>.

3) The subject <record type> of a <key clause> is the member <record type> of the containing <member clause>.

   The contextual <record type> of each <component identifier> in a <key clause> is the subject <record type> of the <key clause>.

4) A <key clause> shall be specified if the containing <set type> specifies <sorted order>; otherwise, a <key clause> shall not be specified.

5) A <key duplicates> shall be specified if the containing <set type> does not specify a <sorted order> with <order duplicates>; otherwise, a <key duplicates> shall not be specified.

   NOTE: *Together with rules governing the <order clause>, this ensures that set members with duplicate key items are controlled either by a single <order duplicates> in the <order clause> or by a <key duplicates> in each <member clause>.*

6) A single <key clause> shall contain exactly one <record type key item> if the containing <set type> specifies a <sorted order> with <record type sequence>; otherwise, a <key clause> shall not contain a <record type key item>.

7) The <component identifier>s and <record type key item> specified in a <key clause> are the *key items* that constitute the *sort control key* for the subject <record type> as a member <record type> of the containing <set type>. If a key item is immediately preceded by ASCENDING or DESCENDING, then the direction of the key item is defined to be respectively *ascending* or *descending*. If a key item K is immediately preceded by another key item K2, then the direction of key item K is defined to be the direction of key item K2.

8) If no <record type key item> is specified in a <key clause>, then all key items in that <key clause> are defined to be common key items; otherwise, the common key items of a <key clause> are defined to be the <record type key item> together with those key items whose ordinal position in the <key clause> is less than the ordinal position of the <record type key item>. Each <member clause> of a <set type> shall contain the same number of common key items. All common key items that are at the same ordinal position in the <key clause> of <member clause>s in a <set type> shall have the same direction and shall either be <record type key item> or shall designate <component type>s that have identical <data type>s.

General Rules

1) When the <component identifier>s of a <key clause> are referenced for a record occurrence, the contextual <database key> for those <component identifier>s is the <database key> of that record occurrence.

2) The key items are stated in a <key clause> in order of decreasing significance.

3) Depending on whether the direction of a key item is *ascending* or *descending*, values of that key item are ordered respectively from the lowest to the highest or from the highest to the lowest.

4) Case:

   a) If a key item is a <component identifier>, then the ordering of that key item is defined by either the *less than* or the *greater than* relation condition, depending on whether the direction of the key item is *ascending* or *descending*.

   b) If a key item is <record type key item>, then the ordering of that key item is defined by the sequence of <record name>s specified in the <record type sequence> of the <sorted order> of the containing <set type>.

5) The <key duplicates> specifies the action to be taken when the execution of a <store statement>, <modify statement>, <connect statement>, or <reconnect statement> would cause an occurrence of the containing <set type> to contain two or more member records that have *equal* values for all of the key items:

   Case:

   a) If PROHIBITED is specified, then the <key clause> is violated.

   b) If LAST or FIRST is specified, then member records that have *equal* values for the key items shall be ordered respectively in chronological or reverse chronological sequence. This chronology is based on the most recent time at which a <store statement>, <modify statement>, <connect statement>, or <reconnect statement> inserted a given member record into the set occurrence or a <modify statement> specified values for one or more key items in the given member record. A <modify statement> that specifies values for one or more key items in a member record that are *equal* to the current values for those key items in that record establishes a new point in the chronology.

   c) If DEFAULT is specified, then member records that have *equal* values for all of the key items shall be ordered in an implementor-defined sequence.

If <key duplicates> is not specified, then the action to be taken is controlled by the <order duplicates> of the containing <set type>.

6.20 <member check clause>

Function

Specify a validity condition on member records of each occurrence of a <set type>.

Format

<member check clause> ::=
CHECK <condition>

Syntax Rules

1) The subject <record type> of a <member check clause> is the member <record type> of the containing <member clause>.

   If the containing <set type> of a <member check clause> is a nonsingular <set type>, then the object <record type> of the <member check clause> is the owner <record type> of the containing <set type>. If the containing <set type> of a <member check clause> is a singular <set type>, then the <member check clause> has no object <record type>.

2) If the subject <record type> and the object <record type> are the same, then every <component identifier> in the <condition> shall contain a <qualifier> that specifies either OWNER or MEMBER.

3) Any <component identifier> in the <condition> that contains a <component name> that is defined in both the subject <record type> and the object <record type> (if any) of the <member check clause> shall contain a <qualifier>.

4) Case:

   a) If a <component identifier> in the <condition> contains a <qualifier> that specifies a <record name>, then that <record name> shall be the <record name> of either the subject <record type> or the object <record type> (if any); the contextual <record type> of the <component identifier> is that <record type>.

   b) If a <component identifier> in the <condition> contains a <qualifier> that specifies OWNER, then the <component identifier> shall have an object <record type>, and the contextual <record type> of the <component identifier> is that <record type>.

   c) If a <component identifier> in the <condition> contains a <qualifier> that specifies MEMBER, then the contextual <record type> of the <component identifier> is the subject <record type>.

   d) If a <component identifier> in the <condition> does not contain a <qualifier>, then the <component name> of that <component identifier> shall be defined in either the subject <record type> or the object <record type> (if any); the contextual <record type> of the <component identifier> is that <record type>.

General Rules

1) When the <condition> of a <member check clause> is evaluated for a set occurrence, the contextual <database key> for the <component identifier>s in that <condition> whose contextual <record type> is the owner <record type> of the set is the <database key> of the owner record occurrence of that set occurrence, and the contextual <database key> for the <component identifier>s in that <condition> whose contextual <record type> is the member <record type> of the set is the <database key> of the member record occurrence of that set occurrence.

2) A <member check clause> is violated if the database contains an occurrence of the containing <set type> in which the <condition> is *false* when evaluated in respect to the owner record occurrence (if any) and any occurrence of the subject <record type> that is a member of that set occurrence.

6.21 <component identifier>

Function

Reference a component or data item.

Format

<component identifier> ::=
<dot style component identifier>
| <of style component identifier>

<dot style component identifier> ::=
[<qualifier>.] <component name> [<subscripts>]

<of style component identifier> ::=
<component name> [<subscripts>] [OF <qualifier>]

<qualifier> ::=
<record name>
| OWNER
| MEMBER

Syntax Rules

1) The *contextual* <record type> of a <component identifier> is specified in the Syntax Rules of the production symbol that immediately contains the <subscripts>.

2) If the <component identifier> contains a <qualifier>, then:

   a) If the <qualifier> contains a <record name>, then that <record name> shall designate the contextual <record type>.

   b) If the <qualifier> specifies OWNER, then the <component identifier> shall be contained in a <set type>, and the owner <record type> of that <set type> shall be the contextual <record type>.

   c) If the <qualifier> specifies MEMBER, then the <component identifier> shall be contained in a <member clause>, and the member <record name> of that <member clause> shall designate the contextual <record type>.

3) The <component name> shall designate a <component type> of the contextual <record type>. That <component type> is the subject <component type>.

4) If a <component identifier> contains <subscripts>, then:

   a) The subject <component type> shall define an array.

   b) The contextual <occurs clause> of the <subscripts> is the <occurs clause> contained in the subject <component type>.

   c) Each <operand> in the <subscripts> shall be a <literal>.

General Rules

1) The contextual <database key> of a <component identifier> is specified in the General Rules of the production symbol that immediately contains the <component identifier>.

2) A <component identifier> references a data item or array in the record referenced by the contextual <database key>, as follows:

   Case:

   a) If the subject <component type> of a <component identifier> defines a data item, then the <component identifier> references that data item.

   b) If the subject <component type> of a <component identifier> defines an array and the <component identifier> does not contain <subscripts>, then the <component identifier> references that array.

   c) If the subject <component type> of a <component identifier> defines an array and the <component identifier> contains <subscripts>, then the <component identifier> references the i-th data item in the array, where i is the value of the <subscripts>.

7. Subschema definition language

7.1 <subschema>

Function

Define a user view of the database.

Format

<subschema> ::=
<subschema name clause>
[{<record view> | <set view>}...]

Syntax Rules

None.

General Rules

None.

7.2 <subschema name clause>

Function

Name a <subschema>.

Format

<subschema name clause> ::=
<dot style subschema name clause>
| <of style subschema name clause>

<dot style subschema name clause> ::=
SUBSCHEMA <schema name>.<subschema name>

<of style subschema name clause> ::=
SUBSCHEMA <subschema name> OF <schema name>

<subschema name> ::= <identifier>

Syntax Rules

1) The <schema name> shall designate a <schema> in the same environment as the <subschema>. That <schema> is the subject <schema> of the <subschema>. The concept of environment is implementor-defined.

2) The <subschema name> of a <subschema> shall be different from the <subschema name> of any other <subschema> that has the same subject <schema>.

General Rules

1) A <subschema name clause> defines the <identifier> to be a <subschema name> that designates the containing <subschema> within the subject <schema>.

7.3 <record view>

Function

Specify that a <record type> is to be included in a <subschema>, and declare an <identifier> that shall designate it within the <subschema>.

Format

<record view> ::=
RECORD [<record renamed>] <record view name>
[<component list>]

<record renamed> ::=
<record name> RENAMED

<record view name> ::=
<identifier>

<component list> ::=
<component view>... | ALL

Syntax Rules

1) Case:

   a) If a <record renamed> is specified, then the <record name> shall designate a <record type> in the subject <schema>. That <record type> is the subject <record type>.

   b) If a <record renamed> is omitted, then the <record view name> shall be a <record name> that designates a <record type> in the subject <schema>. That <record type> is the subject <record type>.

2) The <record view name> shall be different from the <record view name> of every other <record view> in the containing <subschema>. The subject <record type> shall be different from the subject <record type> of every other <record view> in the containing <subschema>.

3) If a <component list> of ALL is specified, then it is equivalent to a <component list> that contains a <component view> for every <component type> in the subject <record type>, where the <component view> corresponding to a <component type> contains a <component view name> that is the <component name> of the <component type>, and contains no <component renamed>.

General Rules

1) A <record view> defines the <identifier> to be a <record view name> that designates the subject <record type>.

7.4 <component view>

Function

Specify that a <component type> is to be included within a <subschema>, and declare an <identifier> that shall designate it within the <subschema>.

Format

<component view> ::=
ITEM [<component renamed>] <component view name>

<component renamed> ::=
<component name> RENAMED

<component view name> ::=
<identifier>

Syntax Rules

1) The subject <record type> is the <record type> designated by the <record view name> of the containing <record view>.

2) Case:

   a) If a <component renamed> is specified, then the <component name> shall designate a <component type> contained in the subject <record type>. That <component type> is the subject <component type>.

   b) If a <component renamed> is omitted, then the <component view name> shall be a <component name> that designates a <component type> contained in the subject <record type>. That <component type> is the subject <component type>.

3) The <component view name> shall be different from the <component view name> of every other <component view> in the containing <record view>. The subject <component type> shall be different from the subject <component type> of every other <component view> in the containing <record view>.

General Rules

1) A <component view> defines the <identifier> to be a <component view name> that designates the subject <component type>.

7.5 <set view>

Function

Specify that a <set type> is to be included in a <subschema>, and declare an <identifier> that shall designate it within the <subschema>.

Format

<set view> ::=
SET [<set renamed>] <set view name>

<set renamed> ::=
<set name> RENAMED

<set view name> ::=
<identifier>

Syntax Rules

1) Case:

   a) If a <set renamed> is specified, then the <set name> shall designate a <set type> in the subject <schema>. That <set type> is the subject <set type>.

   b) If a <set renamed> is omitted, then the <set view name> shall be a <set name> that designates a <set type> in the subject <schema>. That <set type> is the subject <set type>.

2) The <set view name> shall be different from the <set view name> of every other <set view> in the containing <subschema>. The subject <set type> shall be different from the subject <set type> of every other <set view> in the containing <subschema>.

3) The owner <record type> and at least one member <record type> of the subject <set type> shall be designated by a <record view> of the containing <subschema>.

General Rules

1) A <set view> defines the <identifier> to be a <set view name> that designates the subject <set type>.

8. Module language

8.1 <module>

Function

Define a module.

Format

<module> ::=
<module name clause>
<language clause>
<subschema specification>
[<temporary set specifications>]
<procedure>...

<language clause> ::=
LANGUAGE {COBOL | FORTRAN | PASCAL | PLI}

<subschema specification> ::=
<dot style subschema specification>
| <of style subschema specification>

<dot style subschema specification> ::=
SUBSCHEMA <schema name>.<subschema name>

<of style subschema specification> ::=
SUBSCHEMA <subschema name> OF <schema name>

Syntax Rules

1) The <schema name> shall designate a <schema> in the same environment as the <module>. That <schema> is the subject <schema> of the <module>. The concept of environment is implementor-defined.

2) The <subschema name> shall designate a <subschema> in the same environment as the <module>. The subject <schema> of that <subschema> shall be the subject <schema> of the <module>. That <subschema> is the subject <subschema> of the <module>.

General Rules

1) A <module> shall be associated with an application program during its execution. An application program shall be associated with at most one <module>.

2) If the <language clause> of a <module> specifies COBOL (respectively FORTRAN, PASCAL, PLI) and if the agent that performs a call of a <procedure> in that <module> is not a standard COBOL program (respectively standard FORTRAN, Pascal, PL/I program), then the results are undefined.

3) Before the first time that a programming language agent performs a call of a <procedure> in a <module>, construct an initial <session state>, and associate that <session state> with every call by that programming language agent of any <procedure> in the <module>.

4) After the last time that a programming language agent performs a call of a <procedure> in a <module>, perform a <rollback statement> specifying FINISH, and destroy the <session state> associated with the <module>.

8.2 <module name clause>

Function

Name a <module>.

Format

<module name clause> ::=
MODULE [<module name>]

<module name> ::= <identifier>

Syntax Rules

1) The <module name> shall be different from the <module name> of any other <module> in the same environment. The concept of environment is implementor-defined.

General Rules

1) A <module name clause> defines the optional <identifier> to be a <module name> that designates the containing <module> within the environment.

8.3 <temporary set specifications>

Function

Specify temporary sets.

Format

<temporary set specifications> ::=
<temporary set specification>...

<temporary set specification> ::=
SET <set view name>

Syntax Rules

1) The <set view name> of each <temporary set specification> shall be different from the <set view name> of any other <temporary set specification> in the containing <module>, and shall be different from the <set view name> of any <set view> in the subject <subschema>.

2) A <temporary set specification> with a <set view name> S defines a temporary <set type> with an implicit <set name clause>, <owner clause>, and <order clause> as follows:

SET S OWNER SYSTEM ORDER LAST

and, for each <record view name> R specified in the subject <subschema>, a <member clause> with a <member record name clause>, an <insertion clause>, and a <retention clause> as follows:

MEMBER R INSERTION MANUAL RETENTION OPTIONAL

General Rules

1) The <set view name> of a <temporary set specification> designates the temporary <set type> defined by the <temporary set specification>.

8.4 <procedure>

Function

Define a procedure.

Format

```
<procedure> ::=
        PROCEDURE <procedure name> [<parameter declaration>...]
            [<NDL statement>...]

<parameter declaration> ::=
            <parameter name> <data type> [<occurs clause>]
          | RECORD
          | STATUS
          | TEST

<procedure name> ::= <identifier>

<parameter name> ::= <identifier>

<NDL statement> ::=
            <commit statement>
          | <connect statement>
          | <disconnect statement>
          | <erase statement>
          | <find statement>
          | <get statement>
          | <modify statement>
          | <nullify cursor statement>
          | <ready statement>
          | <reconnect statement>
          | <rollback statement>
          | <store statement>
          | <test statement>

<test statement> ::=
            <test database key equal statement>
          | <test database key null statement>
          | <test set empty statement>
          | <test set membership statement>
```

Syntax Rules

1) The <procedure name> shall be different from the <procedure name> of any other <procedure> in the containing <module>.

2) The <parameter name> of each <parameter declaration> in a <procedure> shall be different from the <parameter name> of any other <parameter declaration> in that <procedure>.

3) Any <parameter name> contained in an <NDL statement> in a <procedure> shall be specified in a <parameter declaration> in that <procedure>.

4) If a <component view name> in a <component view identifier> in an <NDL statement> is identical to a <parameter name> specified in the containing <procedure>, then the <component view identifier> shall contain a <record view name>.

5) The <record type> designated by each <record view name> in each <NDL statement> shall be defined in the subject <subschema>.

6) The <set type> designated by each <set view name> in each <NDL statement> shall be defined in the subject <subschema> or in the <temporary set specifications> of the containing <module>.

7) A <procedure> shall include at most one RECORD <parameter declaration>, at most one STATUS <parameter declaration>, and at most one TEST <parameter declaration>.

8) A <procedure> shall contain at most one <test statement>.

9) If a <procedure> contains a <test statement>, then the <procedure> shall contain a TEST <parameter declaration>. If a <procedure> does not contain a <test statement>, then it shall not contain a TEST <parameter declaration>.

10) A <procedure> shall contain at most one <NDL statement> that is a <commit statement> or a <rollback statement>. If a <procedure> contains such an <NDL statement>, then it shall be the last <NDL statement> in the <procedure>.

11) A valid call of a <procedure> shall supply n parameters, where n is the number of <parameter declaration>s in that <procedure>.

12) Case:

    a) If the i-th <parameter declaration> contains an <occurs clause>, then the i-th parameter shall be an array with the same number of data item occurrences.

    b) If the i-th <parameter declaration> is the RECORD <parameter declaration>, then the type of the i-th parameter shall define a single occurrence of a character string of length 18.

    c) If the i-th <parameter declaration> is the STATUS <parameter declaration>, then the type of the i-th parameter shall define a single occurrence of a character string of length 5.

    d) If the i-th <parameter declaration> is the TEST <parameter declaration>, then the type of the i-th parameter shall define a single occurrence of a character string of length 1.

13) If the i-th <parameter declaration> of a <procedure> is respectively the RECORD, STATUS, or TEST <parameter declaration>, then the i-th parameter that is supplied in a call of that <procedure> is referred to as the RECORD parameter, the STATUS parameter, or the TEST parameter.

14) The subject <language clause> of a <procedure> is the <language clause> of the containing <module>.

15) Case:

a) If the subject <language clause> specifies COBOL, then:

i) Any <data type> in a <parameter declaration> shall specify either CHARACTER or NUMERIC.

ii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER L for some <length> L, then the type of the i-th parameter shall be COBOL alphanumeric with a length of L.

iii) If the i-th <parameter declaration> specifies a <data type> that is NUMERIC P S for some <precision> and <scale> P and S, then the type of the i-th parameter shall be COBOL usage DISPLAY sign LEADING SEPARATE and the following PICTURE:

iv) Case:

1. If S>P, then a PICTURE with an "S" followed by a "V" followed by S-P "P"s followed by P "9"s.

2. If S=P, then a PICTURE with an "S" followed by a "V" followed by P "9"s.

3. If P>S>0, then a PICTURE with an "S" followed by P-S "9"s followed by a "V" followed by S "9"s.

4. If S=0, then a PICTURE with an "S" followed by P "9"s followed by a "V".

5. If S<0, then a PICTURE with an "S" followed by P "9"s followed by abs(S) "P"s.

b) If the subject <language clause> specifies FORTRAN, then:

i) Any <data type> in a <parameter declaration> shall specify either CHARACTER, INTEGER, REAL, or DOUBLE PRECISION.

ii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER L for some <length> L, then the type of the i-th parameter shall be FORTRAN CHARACTER with a length of L.

iii) If the i-th <parameter declaration> specifies a <data type> that is INTEGER, REAL, or DOUBLE PRECISION, then the type of the i-th parameter shall be respectively FORTRAN INTEGER, REAL, or DOUBLE PRECISION.

c) If the subject <language clause> specifies PASCAL, then:

i) Any <data type> in a <parameter declaration> shall specify either CHARACTER, INTEGER, or REAL.

ii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER L for some <length> L, then the type of the i-th parameter shall be Pascal string with a length of L.

iii) If the i-th <parameter declaration> specifies a <data type> that is INTEGER or REAL, then the type of the i-th parameter shall be respectively Pascal INTEGER or REAL.

d) If the subject <language clause> specifies PLI, then:

i) Any <data type> in a <parameter declaration> shall specify either CHARACTER, FIXED, or FLOAT.

ii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER L for some <length> L, then the type of the i-th parameter shall be PL/I CHARACTER with a length of L.

iii) If the i-th <parameter declaration> specifies a <data type> that is FIXED P S for some <precision> and <scale> P and S, then the type of the i-th parameter shall be PL/I FIXED REAL DECIMAL (P,S).

iv) If the i-th <parameter declaration> specifies a <data type> that is FLOAT P for some <precision> P, then the type of the i-th parameter shall be PL/I FLOAT REAL BINARY (P).

General Rules

1) A <procedure> defines a procedure that may be called by an implementor-defined agent.

2) When a <procedure> is called by a programming language agent:

   a) If no transaction is active for that agent, then initiate a transaction and associate that transaction with this call and with subsequent calls by that agent of any <procedure> in the containing <module> until the agent terminates that transaction.

   b) If the <procedure> contains a list of one or more <NDL statement>s, then perform each <NDL statement> in that list, from left to right, as specified in the General Rules of each <NDL statement>.

3) The <parameter name> of a <parameter declaration> in a <procedure> designates that <parameter declaration> in the <procedure>.

4) If execution of any <NDL statement> in the <procedure> cannot be completed because of a deadlock, then raise exception *procedure: deadlock*.

5) Case:

   a) If no exception is raised during the execution of any <NDL statement> in the <procedure>, then:

      Case:

      i) If the <procedure> contains a STATUS <parameter declaration>, then set the value of the STATUS parameter to the value specified for *procedure: success* in clause 12, "Status codes" on page 117.

      ii) If the <procedure> contains a RECORD <parameter declaration>, then:

         Case:

         1. If the <database key> of the <session cursor> is not *null*, then set the value of the RECORD parameter to the <record view name> that designates the <record type> of the record referenced by that <database key>.

         2. If the <database key> of the <session cursor> is *null*, then set the value of the RECORD parameter to all space characters.

   b) If an exception is raised during the execution of any <NDL statement> in the <procedure>, then:

      i) Cancel all changes made to the database by the execution of the <procedure>.

ii) Cancel all changes made to the <session state> by the execution of the <procedure>.

iii) If the <procedure> contains a STATUS <parameter declaration>, then set the value of the STATUS parameter to the value specified for the exception in clause 12, "Status codes" on page 117.

iv) If the <procedure> contains a TEST <parameter declaration>, then set the value of the TEST parameter to "0".

9. Data manipulation language

9.1 <commit statement>

Function

Terminate the current transaction with commit.

Format

<commit statement> ::=
COMMIT [FINISH]

Syntax Rules

None.

General Rules

1) Terminate the current transaction.

2) Make all database changes made by the current transaction accessible to concurrent sessions.

3) For each <temporary set> in the <temporary sets>, remove each member record from membership in that set.

4) Set the <cursors> to the initial <cursors> for the subject <subschema>.

5) If FINISH is specified, then set the <ready list> to *empty*.

9.2 <connect statement>

Function

Establish the membership of a record occurrence in a set.

Format

<connect statement> ::=
CONNECT <database key identifier> TO <set view name>

Syntax Rules

1) An eligible <record type> is a <record type> that is defined as a member <record type> of the <set type> designated by the <set view name>, and that has an <insertion clause> specifying MANUAL or a <retention clause> specifying OPTIONAL in that <set type>.

2) If the <database key identifier> specifies a <record view name>, then that <record view name> shall designate an eligible <record type>.

General Rules

1) The object <database key> is the <database key> referenced by the <database key identifier>. If that <database key> is *null*, then raise exception *connect: database key is null.*

   The object <record type> is the <record type> referenced by the object <database key>.

2) If the <record view name> of the object <record type> does not designate a <ready specification> in the <ready list> having an <access intent> of UPDATE, then raise exception *connect: record not ready for update.*

3) If the object <record type> is not an eligible <record type>, then raise exception *connect: ineligible record type.*

4) Perform the <insert operation> with the object <database key> as the <insert record>, the <set view name> of the <connect statement> as the <insert set type>, the <database key> of the <owner> of the <set cursor> designated by the <set view name> of the <connect statement> as the <insert set owner>, and *update* as the <insert cursor disposition>.

9.3 <disconnect statement>

Function

Remove a record from set membership in a specified <set type>.

Format

<disconnect statement> ::=
DISCONNECT <database key identifier> FROM <set view name>

Syntax Rules

1) An eligible <record type> is a <record type> that is defined as a member <record type> of the <set type> designated by the <set view name>, and that has a <retention clause> in that <set type> specifying OPTIONAL.

2) If the <database key identifier> specifies a <record view name>, then that <record view name> shall designate an eligible <record type>.

General Rules

1) The object <database key> is the <database key> referenced by the <database key identifier>. If that <database key> is *null*, then raise exception *disconnect: database key is null*.

   The object <record type> is the <record type> referenced by the object <database key>.

2) If the <record view name> of the object <record type> does not designate a <ready specification> in the <ready list> having an <access intent> of UPDATE, then raise exception *disconnect: record not ready for update*.

3) If the object <record type> is not an eligible <record type>, then raise exception *disconnect: ineligible record type*.

4) Perform the <remove operation> with the object <database key> as the <remove record> and the <set view name> of the <disconnect statement> as the <remove set type>.

9.4 <erase statement>

Function

Remove one or more records from the database.

Format

<erase statement> ::=
ERASE <database key identifier> WITH <cascade specification>

<cascade specification> ::=
FULL CASCADE | PARTIAL CASCADE

Syntax Rules

None.

General Rules

1) The object <database key> is the <database key> referenced by the <database key identifier>. If that <database key> is *null*, then raise exception *erase: database key is null*.

   The object record is the record referenced by the object <database key>. The object <record type> is the <record type> of the object record.

2) If the <record view name> of the object <record type> does not designate a <ready specification> in the <ready list> having an <access intent> of UPDATE, then raise exception *erase: record not ready for update*.

3) For each <set type> of which the object record is currently a member, perform the <remove operation> with the object <database key> as the <remove record> and the <set name> of the <set type> as the <remove set type>.

4) The affected sets are those whose owner record is the object record. The affected records are those that are members of one or more affected sets.

5) Case:

   a) If the <cascade specification> specifies FULL CASCADE, then erase all affected records.

   b) If the <cascade specification> specifies PARTIAL CASCADE, then:

      i) If any affected set contains any mandatory member, then raise exception *erase: set has mandatory member*.

      ii) Erase any affected record that is a fixed member of any affected set.

      iii) For each affected record that is an optional member of any affected set, perform the <remove operation> with the <database key> of the affected record as the <remove record> and the <set name> of the <set type> of the affected set as the <remove set type>.

6) Erase the object record. The relative order of other record occurrences under implementor-defined ordering criteria remains unchanged.

7) If the <database key> of the <session cursor> is equal to the <database key> of the object record, then set the <session cursor> to *null*.

   If the <database key> of any <record cursor> is equal to the <database key> of the object record, then set the <database key> of that <record cursor> to *null*.

   If the <database key> of the object record is equal to the <owner> in any <set cursor>, then set both the <owner> and the <position> of that <set cursor> to *null*.

8) Apply General Rules 3 through 8 of the <erase statement> to each erased record as if it were the object record.

9.5 <find statement>

Function

Select a record in the database.

Format

```
<find statement> ::=
        FIND
            <find specification>
            [<find intent>]
            [<find cursor disposition>]

<find specification> ::=
            <database key identifier>
          | <search specification>

<search specification> ::=
        <search orientation>
        <domain specification>
        [WHERE <condition>]

<search orientation> ::=
        FIRST | LAST | NEXT | PRIOR
        | {ABSOLUTE | RELATIVE} <signed integer>

<domain specification> ::=
            <record type domain>
          | <set domain>
          | <subschema domain>

<record type domain> ::=
        <record view name>

<set domain> ::=
        [<record view name>] IN <set view name>

<subschema domain> ::=
        SUBSCHEMA RECORD

<find intent> ::=
        FOR {RETRIEVE | UPDATE}

<find cursor disposition> ::=
            RETAIN ALL
          | <find specific disposition>

<find specific disposition> ::=
        [<position member>]
        [<find specific retention>]
```

<position member> ::=
AS MEMBER <set view name>...

<find specific retention> ::=
RETAIN RECORD
| RETAIN SET <set view name>...
| RETAIN RECORD SET <set view name>...

Syntax Rules

1) If the <find specification> is a <search specification>, then:

   a) If the <search orientation> contains a <signed integer>, then the value of that <signed integer> shall not be 0.

   b) Case:

      i) If the <domain specification> is a <record type domain>, then the object <record type> of the <search specification> is the <record type> designated by the <record view name> contained in the <domain specification>.

      ii) If the <domain specification> is a <set domain>, then:

         1. The object <set type> of the <domain specification> is the <set type> designated by the <set view name>.

         2. If the <search specification> includes a WHERE <condition> and the object <set type> contains multiple member <record type>s, then the <set domain> shall specify a <record view name>.

            If the <set domain> specifies a <record view name>, then the <record type> designated by <record view name> shall be a member <record type> of the object <set type>. That <record type> is the object <record type> of the <search specification>.

            If the <set domain> does not specify a <record view name>, then the object <record type>s of the <search specification> are all subschema <record type>s that are member <record type>s of the object <set type>.

      iii) If the <domain specification> is a <subschema domain>, then:

         1. The object <record type>s of the <search specification> are the <record type>s specified by the <record view>s of the subject <subschema>.

         2. If the <search specification> includes a WHERE <condition>, then the subject <subschema> shall contain exactly one <record view>.

   c) If the <search specification> contains a WHERE <condition>, then the contextual <record type> of each <component view identifier> in the <condition> is the object <record type>

2) Every <set view name> contained within a <find cursor disposition> shall be different from every other <set view name> within that <find cursor disposition>.

3) A <set view name> contained in a <position member> shall designate a <set type> that has a member <record type> identical to its owner <record type>.

4) If the <find intent> is omitted, then it is assumed to be FOR RETRIEVE.

General Rules

1) Locate a specific record occurrence in the database, determined by the <find specification> as follows:

Case:

a) If the <find specification> is a <database key identifier>, then:

i) The object <database key> is the <database key> referenced by the <database key identifier>.

ii) If the object <database key> is *null*, then raise exception *find: database key is null*.

iii) Select the record referenced by the object <database key>.

b) If the <find specification> is a <search specification>, then:

i) Case:

1. If the <domain specification> is a <record type domain>, then the domain of the <search specification> consists of all record occurrences of the object <record type> of the <search specification>. The ordering of the record occurrences in the domain is implementor-defined. This order is subject to the reproducibility requirement within a transaction, but it may change between transactions. The object-position of the <search specification> is the <database key> of the <record cursor> designated by the <record view name>.

2. If the <domain specification> is a <set domain>, then:

A. The object <set cursor> of the <set domain> is the <set cursor> designated by the <set view name>.

B. If the object <set type> is singular, then the object set of the <set domain> is the one and only occurrence of the object <set type> in the database.

If the object <set type> is not singular, then the object set of the <set domain> is the set referenced by the <database key> of the <owner> of the object <set cursor>. If that <database key> is *null*, then raise exception *find: database key is null*.

C. The domain of the <search specification> consists of all record occurrences that are members of the object set. The ordering of the record occurrences in the domain is determined by the ordering criteria for the object <set type>.

D. The object-position of the <search specification> is the <position> of the object <set cursor>.

3. If the <domain specification> is a <subschema domain>, then the domain of the <search specification> consists of all record occurrences of the object <record type>s. The ordering of record occurrences in the domain is implementor-defined. This order is subject to the reproducibility requirement within a transaction, but it may change between transactions.

The object-position of the <search specification> is the <database key> of the <session cursor>.

ii) If the <search orientation> contains a <signed integer>, then let i be the value of that <signed integer>. If the <search orientation> specifies FIRST or NEXT, then let i be +1. If the <search orientation> specifies LAST or PRIOR, then let i be -1.

iii) If NEXT, PRIOR, or RELATIVE is specified and the value of i is positive (negative), then:

Case:

1. If the object-position is a single <database key> that is not *null*, then remove from the domain the record referenced by that <database key> and all record occurrences that are ordered before (after) that record.

2. If the object-position is a pair of <database key>s whose second (first) <database key> is not *null*, then remove from the domain all record occurrences that are ordered before (after) the record referenced by that <database key>.

3. If the object-position is a pair of <database key>s whose second (first) <database key> is *null*, then remove all record occurrences from the domain.

iv) If a <set domain> containing a <record view name> is specified, then remove from the domain all record occurrences that are not of the object <record type>.

v) If the <search specification> contains a <condition>, then for each record occurrence in the domain:

1. The contextual <database key> of each <component view identifier> in the <condition> is the <database key> of the record occurrence.

2. Evaluate that <condition> for the record occurrence.

3. If the <condition> is *false*, then remove the record occurrence from the domain.

vi) Let n be the number of record occurrences remaining in the domain. If the value of i is positive, then let j be i. Otherwise, let j be n+i+1.

If j is greater than 0 and not greater than n, then select the j-th record occurrence of the domain. Otherwise, raise exception *find: no record found.*

2) If the <record view name> of the selected record does not designate a <ready specification> in the <ready list>, then raise exception *find: record not ready.* If the <access intent> of that <ready specification> is RETRIEVE and the <find intent> specifies UPDATE, then raise exception *find: record not ready for update.*

3) Set the <database key> of the <session cursor> to the <database key> of the selected record.

4) If the <find cursor disposition> is omitted or it specifies neither ALL nor RECORD, then set the <database key> of the <record cursor> designated by the <record view name> of the selected record to the <database key> of the selected record.

5) If the <find cursor disposition> is omitted or it specifies does not specify ALL, then for each <set cursor> whose <set view name> is not specified in a <find specific retention> in the <find cursor disposition>:

Case:

a) If the <record type> of the selected record is the owner <record type> of the <set type> of the <set cursor>, and if the <set view name> that designates the <set cursor> is not specified in a <position member>, then set the <owner> of that <set cursor> to the <database key> of the selected record, and set the <position> of that <set cursor> to *null*.

b) If the <set view name> of the <set cursor> is specified in a <position member> and the selected record is not a member of any set occurrence of the <set type> designated by that <set view name>, then raise exception *find: no record found*.

c) If the selected record is a member of a set occurrence of the <set type> designated by the <set view name> of the <set cursor>, then if either the <record type> of the selected record is not the owner <record type> of that <set type> or the <set view name> of the <set cursor> is specified in a <position member>, then:

   i) Set the <position> of the <set cursor> to the <database key> of the selected record.

   ii) If that <set type> is a nonsingular <set type>, then set the <owner> of the <set cursor> to the <database key> of the owner record occurrence of that set occurrence.

d) Otherwise, do not update the <set cursor>.

9.6 <get statement>

Function

Set parameter values to component values from a selected record occurrence.

Format

<get statement> ::=
GET <to parameter move>

Syntax Rules

None.

General Rules

1) The object record is the record referenced by the <database key> of the <record cursor> designated by the <record view name> of the <to parameter move>. If that <database key> is *null*, then raise exception *get: record cursor is null.*

2) The contextual <database key> of each <component view identifier> in the <to parameter move clause> is the <database key> of the object record.

3) Evaluate all <operand>s in the <to parameter move>.

4) For each <to parameter move clause> in the <to parameter move>, from left to right:

   a) Perform the data transfer specified by the <to parameter move clause>.

   b) If an exception is raised while performing the data transfer for the <to parameter move clause>, then do not assign a value to the parameter designated by that <to parameter move clause>, do not perform any subsequent <to parameter move clause> of the <to parameter move>, and retain the effect of prior <to parameter move clause>s that have been performed.

5) Set the <database key> of the <session cursor> to the <database key> of the object record.

9.7 <modify statement>

Function

Replace the contents of one or more data items in a record occurrence.

Format

<modify statement> ::=
MODIFY <to database move>

Syntax Rules

None.

General Rules

1) The object <record view name> is the <record view name> of the <to database move>. The object <record type> is the <record type> designated by the object <record view name>.

2) If the object <record view name> does not designate a <ready specification> in the <ready list> having an <access intent> of UPDATE, then raise exception *modify: record not ready for update.*

3) The object record is the record referenced by the <database key> of the <record cursor> designated by the object <record view name>. If that <database key> is *null*, then raise exception *modify: record cursor is null.*

4) A modified data item is any data item designated by a <component view identifier> in a target identifier of the <to database move>.

5) The contextual <database key> of each <component view identifier> in the <to database move> is the <database key> of the object record.

6) Evaluate all <operand>s in the <to database move>.

7) For each <to database move clause> in the <to database move>, perform the data transfer specified by each <to database move clause> in the <to database move>.

8) For each <set type> of which the object <record type> is a member <record type>, let the object <member clause> be the <member clause> of that <set type> whose member <record type> is the object <record type>. If the <insertion clause> of the object <member clause> specifies STRUCTURAL, and if a modified data item is referenced by a <component identifier> that is specified as a <member component identifier> in that <insertion clause>, then:

   a) Perform the <remove operation> with the <database key> of the object record as the <remove record>, and the <set view name> of the <set type> as the <remove set type>.

   b) Let D be the <database key> of the occurrence of the <record type> designated by the owner <record view name> of the <set type> that has values for each data item referenced by an <owner component identifier> of a <component identifier match> of the <insertion clause> *equal* to the value of the data item in the object record that is referenced by the <member component identifier>

of that <component identifier match>. If there is no such record occurrence, then raise exception *modify: no match for set insertion.*

c) Perform the <insert operation> with the <database key> of the object record as the <insert record>, the <set name> of the <set type> as the <insert set type>, the <database key> D as the <insert set owner>, and *retain* as the <insert cursor disposition>.

d) If the <retention clause> of the object <member clause> specifies FIXED and the set occurrence into which the object record was inserted is not the set occurrence from which it was removed, then raise exception *modify: retention is fixed.*

9) If the object record is currently a member of a set of a <set type> for which the preceding General Rule does not apply, and a modified component is designated by a <component identifier> in a <key clause> of the <set type>, then:

a) Perform the <remove operation> with the <database key> of the object record as the <remove record> and the <set name> of the <set type> as the <remove set type>.

b) Perform the <insert operation> with the <database key> of the object record as the <insert record>, the <set name> of the <set type> as the <insert set type>, *retain* as the <insert cursor disposition>, and:

Case:

i) If the <set type> is a singular <set type>, then *null* as the <insert set owner>.

ii) If the <set type> is not a singular <set type>, then the <database key> of the owner of the set from which the object record was removed as the <insert set owner>.

10) If execution of the <modify statement> would cause a <record uniqueness clause>, <member uniqueness clause>, <key clause>, or <order clause> to be violated, then raise exception *modify: duplicates are prohibited.* If it would cause a <record check clause> to be violated, then raise exception *modify: record check violated.* If it would cause a <member check clause> to be violated, then raise exception *modify: member check violated.*

9.8 <nullify cursor statement>

Function

Set the referenced cursor to *null*.

Format

<nullify cursor statement> ::=
NULLIFY <database key identifier>

Syntax Rules

None.

General Rules

1) Case:

a) If the <database key identifier> specifies SESSION, then set the <database key> of the <session cursor> to *null*.

b) If the <database key identifier> specifies a <record view name>, then set the <database key> of the <record cursor> designated by the <record view name> of the <nullify cursor statement> to *null*.

c) If the <database key identifier> specifies OWNER, then set both the <owner> and the <position> of the <set cursor> designated by the <set view name> of the <nullify cursor statement> to *null*.

d) If the <database key identifier> specifies MEMBER, then set the <position> of the <set cursor> designated by the <set view name> of the <nullify cursor statement> to *null*.

9.9 <ready statement>

Function

Prepare one or more <record type>s for processing.

Format

<ready statement> ::=
READY <ready specification>...

<ready specification> ::=
<record view name>
<share specification>
<access intent>

<share specification> ::=
EXCLUSIVE | PROTECTED | SHARED

<access intent> ::=
RETRIEVE | UPDATE

Syntax Rules

1) The same <record view name> shall not be specified more than once in a <ready statement>.

General Rules

1) If a <record view name> in the <ready statement> designates a <ready specification> in the <ready list>, then raise exception *ready: record already ready.*

2) If any of the following conditions is *true*, then raise exception *ready: lock conflict.*

   a) A <ready specification> has a <share specification> that is EXCLUSIVE and a <record view name> that is contained in the <ready list> of a concurrent <session state>.

   b) One or more of the <record view name>s is contained in the <ready list> of a concurrent <session state> with a <share specification> that is EXCLUSIVE.

   c) A <ready specification> has a <share specification> that is SHARED, an <access intent> that is UPDATE, and a <record view name> that is specified in the <ready list> of a concurrent <session state> with a <share specification> that is PROTECTED.

   d) A <ready specification> has a <share specification> that is PROTECTED, an <access intent> that is UPDATE, and a <record view name> that is specified in the <ready list> of a concurrent <session state> with a <share specification> of PROTECTED or with an <access intent> of UPDATE.

   e) A <ready specification> has a <share specification> that is PROTECTED, an <access intent> that is RETRIEVE, and a <record view name> that is specified in the <ready list> of a concurrent <session state> with an <access intent> of UPDATE.

3) Append the <ready specification>s to the <ready list> of the <session state>.

4) The <record view name> of a <ready specification> designates that <ready specification>.

9.10 <reconnect statement>

Function

Change the membership of a record occurrence in a <set type>

Format

<reconnect statement> ::=
RECONNECT <database key identifier> IN <set view name>

Syntax Rules

1) An eligible <record type> is a <record type> that is defined as a member <record type> in the <set type> designated by the <set view name>.

2) If the <database key identifier> specifies a <record view name>, then that <record view name> shall designate an eligible <record type>.

General Rules

1) The object <database key> is the <database key> referenced by the <database key identifier>. If that <database key> is *null*, then raise exception *reconnect: database key is null.*

   The object <record type> is the <record type> referenced by the object <database key>.

2) If the <record view name> of the object <record type> does not designate a <ready specification> in the <ready list> having an <access intent> of UPDATE, then raise exception *reconnect: record not ready for update.*

3) If the object <record type> is not an eligible <record type>, then raise exception *reconnect: ineligible record type.*

4) Perform the <remove operation> with the object <database key> as the <remove record> and the <set view name> of the <reconnect statement> as the <remove set type>.

5) Perform the <insert operation> with the object <database key> as the <insert record>, the <set view name> of the <reconnect statement> as the <insert set type>, the <database key> of the <owner> of the <set cursor> designated by the <set view name> of the <reconnect statement> as the <insert set owner>, and *update* as the <insert cursor disposition>.

6) If the object <record type> has a <retention clause> specifying FIXED in the <set type> designated by the <set view name>, and the set into which the object record was inserted is not the set from which it was removed, then raise exception *reconnect: retention is fixed.*

9.11 <rollback statement>

Function

Terminate the current transaction with rollback.

Format

<rollback statement> ::=
ROLLBACK [FINISH]

Syntax Rules

None.

General Rules

1) Terminate the current transaction. Cancel the effect on the database of all statements executed by the session during that transaction.

2) Set the <cursors> to the initial <cursors> for the subject <subschema>.

3) For each <temporary set> in the <temporary sets>, remove each member record from membership in that set.

4) If FINISH is specified, then set the <ready list> to *empty*.

9.12 <store statement>

Function

Store a record in the database.

Format

<store statement> ::=
STORE <to database move> [<store retention>]

<store retention> ::=
RETAIN ALL
| RETAIN RECORD
| RETAIN SET <set view name>...
| RETAIN RECORD SET <set view name>...

Syntax Rules

1) The object <record view name> is the <record view name> of the <to database move>. The object <record type> is the <record type> designated by the object <record view name>. The object <record view> is the <record view> designated by the object <record view name>.

2) Each <component type> in the object <record type> that is not designated by a <component view name> in the object <record view> shall contain a <default clause>.

3) Each <component type> in the object <record type> that is not designated by the <component view name> of a target <component view identifier> in the <to database move> shall contain a <default clause>.

4) The subject <subschema> shall include a <set view> for each <set type> in which the object <record type> is defined as a member <record type> with an <insertion clause> that specifies AUTOMATIC.

5) Every <set view name> contained within a <store retention> shall be different from every other <set view name> within that <store retention>.

6) For each <set view name> specified in a <store retention>, the object <record type> shall be either a member <record type> or owner <record type> of the <set type> referenced by that <set view name>.

General Rules

1) If the object <record view name> does not designate a <ready specification> in the <ready list> having an <access intent> of UPDATE, then raise exception *store: record not ready for update.*

2) Create in the database a record occurrence of the object <record type>. The new record occurrence is the object record. The object record assumes an implementor-defined position relative to other record occurrences, and the relative order of the other occurrences remains unchanged.

NOTE: *The construction of a new record occurrence is specified in the General Rules of 6.8, "<default clause>" on page 38.*

3) The contextual <database key> of each <component view identifier> in the <to database move clause> is the <database key> of the object record.

4) Evaluate all <operand>s in the <to database move>.

5) For each <to database move clause> in the <to database move>, perform the data transfer specified by each <to database move clause> in the <to database move>.

6) Establish the object record as the owner of an empty set for each <set type> in which the object <record type> is the owner <record type>.

7) For each <set type> of which the object <record type> is a member <record type>:

   a) Let the object <member clause> be the <member clause> of that <set type> whose member <record type> is the object <record type>.

   b) If the <store retention> specifies ALL, or if the <set type> is designated by a <set view name> in the <store retention>, then let C be *retain*. Otherwise, let C be *update*.

   c) Case:

      i) If the <insertion clause> of the object <member clause> specifies AUTOMATIC, then perform the <insert operation> with the <database key> of the object record as the <insert record>, the <set name> of the <set type> as the <insert set type>, the <database key> of the <owner> of the <set cursor> whose <set view name> designates the <set type> as the <insert set owner>, and C as the <insert cursor disposition>.

      ii) If the <insertion clause> of the object <member clause> specifies STRUCTURAL, then:

         1. Let D be the <database key> of the occurrence of the owner <record type> of the <set type> that has values for each data item referenced by an <owner component identifier> of a <component identifier match> of the <insertion clause> *equal* to the value of the data item in the record occurrence being stored that is referenced by the <member component identifier> of that <component identifier match>. If there is no such record occurrence, then raise exception *store: no match for set insertion*.

         2. Perform the <insert operation> with the <database key> of the object record as the <insert record>, the <set name> of the <set type> as the <insert set type>, the <database key> D as the <insert set owner>, and C as the <insert cursor disposition>.

      iii) If the <insertion clause> of the object <member clause> specifies MANUAL, then do not insert the object record in any occurrence of the <set type>.

8) Set the <session cursor> to reference the object record.

9) If the <store retention> specifies neither ALL nor RECORD, then set the <record cursor> designated by <record view name> to reference the object record.

   If the <store retention> does not specify ALL, then for each <set type> designated by a <set view> in the subject <subschema> in which the object <record type> is the owner <record type>, if that <set type> is not designated by a <set view name> in the <store retention>, then set the <owner> of the <set cursor> for that <set type> to the <database key> of the object record and set the <position> of that <set cursor> to *null*.

10) If execution of the <store statement> would cause a <record uniqueness clause> to be violated, then raise exception *store: duplicates are prohibited.* If it would cause a <record check clause> to be violated, then raise exception *store: record check violated.*

9.13 <test database key equal statement>

Function

Determine whether two <database key>s reference the same record.

Format

<test database key equal statement> ::=
TEST <operand 1> = <operand 2>

<operand 1> ::= <database key identifier>

<operand 2> ::= <database key identifier>

Syntax Rules

None.

General Rules

1) If the <database key> referenced by <operand 1> or <operand 2> is *null*, then raise exception *test: database key is null.*

2) If the <database key> referenced by <operand 1> and the <database key> referenced by <operand 2> reference the same record occurrence, then set the value of the TEST parameter to "1". Otherwise, set the value of the TEST parameter to "0".

9.14 <test database key null statement>

Function

Determine whether a <database key> is *null*.

Format

<test database key null statement> ::=
TEST NULL <database key identifier>

Syntax Rules

None

General Rules

1) If the <database key> referenced by the <database key identifier> is *null*, then set the value of the TEST parameter to "1". Otherwise, set the value of the TEST parameter to "0".

9.15 <test set empty statement>

Function

Determine whether a set has any member records.

Format

<test set empty statement> ::=
TEST SET EMPTY <set view name>

Syntax Rules

1) The object <set type> is the <set type> designated by <set view name>. The object <set cursor> is the <set cursor> designated by <set view name>.

General Rules

1) Case:

   a) If the object <set type> is a singular <set type>, then the object set is the one-and-only occurrence of the object <set type>.

   b) If the object <set type> is a nonsingular <set type>, then:

      Case:

      i) If the <owner> of the object <set cursor> is *null*, then raise exception *test: set cursor is null*.

      ii) If the <owner> of the object <set cursor> is not *null*, then the object set is the set referenced by the object <set cursor>.

2) If the object set has one or more members whose <record type> is included in the subject <subschema>, then set the value of the TEST parameter to "0". Otherwise, set the value of the TEST parameter to "1".

9.16 <test set membership statement>

Function

Determine whether a record is a member of some occurrence of a <set type>.

Format

<test set membership statement> ::=
TEST SET <set view name> CONTAINS <database key identifier>

Syntax Rules

None.

General Rules

1) The object <database key> is the <database key> referenced by the <database key identifier>. If the object <database key> is *null*, then raise exception *test: database key is null*.

2) If the <record type> of the record referenced by the object <database key> is not defined as a member <record type> of the <set type> designated by <set view name>, then raise exception *test: ineligible record type*.

3) If the record referenced by the object <database key> is a member of an occurrence of the <set type> designated by <set view name>, then set the value of the TEST parameter to "1". Otherwise, set the value of the TEST parameter to "0".

9.17 <database key identifier>

Function

Reference a <database key>.

Format

```
<database key identifier> ::=
        SESSION
      | <record view name>
      | {OWNER | MEMBER} <set view name>
```

Syntax Rules

1) If the <database key identifier> specifies OWNER, then the <set type> designated by the <set view name> shall not be a singular <set type>.

General Rules

1) Case:

   a) If SESSION is specified, then the <database key identifier> references the <database key> of the <session cursor>.

   b) If a <record view name> is specified, then the <database key identifier> references the <database key> of the <record cursor> designated by the <record view name> of the <database key identifier>.

   c) If OWNER is specified, then the <database key identifier> references the <database key> of the <owner> of the <set cursor> designated by the <set view name> of the <database key identifier>.

   d) If MEMBER is specified, then

      i) The object <position> is the <position> of the <set cursor> designated by the <set view name> of the <database key identifier>.

      ii) If the object <position> is a single <database key>, then the <database key identifier> references that <database key>.

      If the object <position> is a pair of <database key>s, then the <database key identifier> references *null*.

9.18 <component view identifier>

Function

Reference a component or data item.

Format

<component view identifier> ::=
<dot style component view identifier>
| <of style component view identifier>

<dot style component view identifier> ::=
[<record view name>.] <component view name> [<subscripts>] [CURSOR]

<of style component view identifier> ::=
<component view name> [<subscripts>]
[OF <record view name>] [CURSOR]

Syntax Rules

1) The contextual <record type> of a <component view identifier> is specified in the Syntax Rules of the <NDL statement> that contains the <component view identifier>.

2) Case:

   a) If a <component view identifier> does not specify CURSOR, then the <component view identifier> shall designate a <component type> of the contextual <record type>, and the <record view name>, if specified, shall designate the contextual <record type>. The subject <record type> of the <component view identifier> is the contextual <record type>.

   b) If a <component view identifier> specifies CURSOR, then it shall specify a <record view name>, and the subject <record view> is the <record view> designated by that <record view name>.

3) The <component view name> shall designate a <component type> of the subject <record type>. That <component type> is the subject <component type>.

4) If a <component view identifier> contains <subscripts>, then:

   a) The subject <component type> shall define an array.

   b) The contextual <occurs clause> of the <subscripts> is the <occurs clause> contained in the subject <component type>.

General Rules

1) The contextual <database key> of a <component view identifier> is specified in the General Rules of the statement that contains the <component view identifier>.

2) Case:

a) If a <component view identifier> specifies CURSOR, then the subject <database key> is the <database key> of the <record cursor> whose <record view name> designates the subject <record type>. If that <database key> is *null*, then raise exception *cursor reference: database key is null.*

b) If a <component view identifier> does not specify CURSOR, then the subject <database key> is the contextual <database key>.

3) A <component view identifier> references a data item or array in the record referenced by the subject <database key>, as follows:

Case:

a) If the subject <component type> of a <component view identifier> defines a data item, then the <component view identifier> references that data item.

b) If the subject <component type> of a <component view identifier> defines an array and the <component view identifier> does not contain <subscripts>, then the <component view identifier> references that array.

c) If the subject <component type> of a <component view identifier> defines an array and the <component view identifier> contains <subscripts>, then the <component view identifier> references the i-th data item in the array, where i is the value of the <subscripts>.

9.19 <parameter identifier>

Function

Reference a parameter item.

Format

<parameter identifier> ::=
<parameter name> [<subscripts>]

Syntax Rules

1) The object <parameter declaration> of a <parameter identifier> is the <parameter declaration> designated by the <parameter name> of the <parameter identifier>.

2) If the object <parameter declaration> does not contain an <occurs clause>, then <subscripts> shall not be specified.

   If the object <parameter declaration> contains an <occurs clause> and a <subscripts> is specified, then that <occurs clause> is the contextual <occurs clause> of the <subscripts>.

General Rules

1) Case:

   a) If a <subscripts> is not specified, then the <parameter identifier> references the data item or array supplied as the parameter corresponding with the object <parameter declaration>.

   b) If a <subscripts> is specified, then the <parameter identifier> references the i-th data item in the array supplied as the parameter corresponding with the object <parameter declaration>, where i is the value of the <subscripts>.

9.20 <to parameter move> and <to database move>

Format

<to parameter move> ::=
<record view name> <to parameter move clause>...

<to parameter move clause> ::=
SET <parameter identifier> TO <operand>

<to database move> ::=
<record view name> [<to database move clause>...]

<to database move clause> ::=
SET <component view identifier> TO <operand>

Syntax Rules

1) For a <to parameter move>, the target identifier is the <parameter identifier>. For a <to database move> the target identifier is the <component view identifier>.

2) Let a *move* be either a <to parameter move> or a <to database move>. Let a *move clause* be either a <to parameter move clause> or a <to database move clause>.

3) The object <record type> of the move and the contextual <record type> of each <component view identifier> in the move is the <record type> designated by the <record view name>.

4) Case:

   a) If the type of the target identifier of a move clause is character string, then the type of the source <operand> shall be character string.

   b) If the type of the target identifier of a move clause is exact numeric, then the type of the source <operand> shall be exact numeric.

   c) If the type of the target identifier of a move clause is approximate numeric, then the type of the source <operand> shall be approximate numeric or exact numeric.

5) Case:

   a) If the target identifier of a move clause references a data item or a data item of an array, then the <operand> of that move clause shall be a data item.

   b) If the target identifier of a move clause references an array, then the <operand> of that move clause shall be an array.

6) No <component view identifier> that is a target identifier shall specify CURSOR. If a <component view identifier> that is a target identifer specifies a <record view name>, then that <record view name> shall be the same as the <record view name> of the <to database move>.

7) A <component view identifier> that is contained in an <operand> of the <to database move> shall specify CURSOR.

General Rules

1) The target item is the item referenced by the target identifier. The source value is the value of the source <operand>.

2) Case:

   a) If the type of the target item is character string of length L, then:

      Case:

      i) If the length of the source value is L, then set the target item to the source value.

      ii) If the length of the source value is M, and M>L, then if the last M-L characters of the source value are space characters, then set the target item to the first L characters of the source value; otherwise, raise exception *data transfer: string truncation*.

      iii) If the length of the source value is M, and M<L, then set the first M characters of the target item to the source value, and set the last L-M characters of the target item to space characters.

   b) If the type of the target item is exact numeric and the source value can be represented exactly as a value of the target item data type, then set the target item to that value; otherwise, raise exception *data transfer: numeric truncation*.

   c) If the type of the target item is approximate numeric, then set the target item to the approximate numeric value of the source value.

10. Auxiliary operations

10.1 <insert operation>

Function

Insert a record occurrence into a set.

Format

<insert operation> ::=
<insert record>
<insert set type>
<insert set owner>
<insert cursor disposition>

<insert record> ::=
<database key>

<insert set type> ::=
<set name> | <set view name>

<insert set owner> ::=
<database key>

<insert cursor disposition> ::=
retain | *update*

Syntax Rules

None.

General Rules

1) The object record is the record referenced by the <database key> of the <insert record>. The object <record type> is the <record type> of the object record. The object <set type> is the <set type> designated by the <set name> or <set view name> of the <insert set type>. The object <set cursor>, if any, is the <set cursor>, if any, whose <set view name> designates the object <set type>.

2) If the object record is already a member of an occurrence of the object <set type>, then raise exception *insert: record already a member of set*.

3) Case:

   a) If the object <set type> is a singular <set type>, then the object set is the one-and-only occurrence of the object <set type>.

   b) If the object <set type> is not a singular <set type>, then the object set is the occurrence of the object <set type> that is owned by the record occurrence referenced by the <database key> of the

<insert set owner>. If that <database key> is *null*, then raise exception *insert: set cursor is null.* If the <order clause> of the object <set type> specifies NEXT or PRIOR, then:

i) If there is no object <set cursor>, then raise exception *insert: set is not in subschema.*

ii) If the object set is not the set occurrence referenced by the object <set cursor>, then raise exception *insert: record not member of set.*

4) Insert the object record as a member of the object set in accordance with the set ordering criteria specified by the <order clause> or the <key clause>s of the object <set type>. The relative order of other members of the object set remains unchanged.

5) If the <order clause> of the object <set type> specifies NEXT (PRIOR), then:

Case:

a) If the <position> of the object <set cursor> is a single <database key> that is not *null*, then insert the object record as a member record in the object set immediately after (before) the member record referenced by that <database key>.

b) If the <position> of the object <set cursor> is a single <database key> that is *null*, then insert the object record as the first (last) member record in the set occurrence referenced by that <set cursor>.

c) If the <position> of the object <set cursor> is a pair of <database key>s and the first (second) of those <database key>s is not *null*, then insert the object record immediately after (before) the member record referenced by that first (second) <database key>.

d) If the <position> of the object <set cursor> is a pair of <database key>s and the first (second) of those <database key>s is *null*, then insert the object record as the first (last) member record in the set occurrence referenced by that <set cursor>.

6) If the <insert operation> would cause a <member uniqueness clause>, an <order clause>, or a <key clause> in the object <set type> to be violated, then raise exception *insert: duplicates are prohibited.* If it would cause a <member check clause> or an implied <member check clause> derived from the <insertion clause> to be violated, then raise exception *insert: member check violated.*

7) If there is an object <set cursor>, then:

Case:

a) If the <insert cursor disposition> is *update*, then set the <position> of the object <set cursor> to the <database key> of the object record.

b) If the <insert cursor disposition> is *retain* and the following condition is *true*, then set the <position> of the object <set cursor> to the <database key> of the object record.

Condition: The <position> of the object <set cursor> is a pair of <database key>s such that either the first such <database key> references a record that immediately precedes the object record in the object set or the second such <database key> references a record that immediately follows the object record in the object set.

10.2 <remove operation>

Function

Remove a record occurrence from a set.

Format

<remove operation> ::=
<remove record>
<remove set type>

<remove record> ::=
<database key>

<remove set type> ::=
<set name> | <set view name>

Syntax Rules

None.

General Rules

1) The object record is the record referenced by the <database key> of the <remove record>. The object <record type> is the <record type> of the object record. The object <set type> is the <set type> designated by the <set name> or <set view name> of the <remove set type>.

2) The object set is the occurrence of the object <set type> of which the object record is a member. If the object record is not currently a member of any occurrence of the object <set type>, then raise exception *remove: record not member of set*.

3) If there is a <set cursor> whose <set view name> designates the object <set type>, then:

   Case:

   a) If the <position> of that <set cursor> is a single <database key> that references the object record, then:

      i) If there is a record occurrence that is a member of the object set and that immediately precedes the object record in that set, then let L be the <database key> of that record occurrence. Otherwise, let L be *null*.

      ii) If there is a record occurrence that is a member of the object set and that immediately follows the object record in that set, then let R be the <database key> of that record occurrence. Otherwise, let R be *null*.

      iii) Set the <position> of that <set cursor> to the pair of <database key>s L and R, in that order.

   b) If the <position> of that <set cursor> is a pair of <database key>s the first (second) of which references the object record, then: