INTERNATIONAL STANDARD

ISO/IEC 9075-5

First edition
1999-12-01

Information technology — Database languages — SQL — Part 5: Host Language Bindings (SQL/Bindings)

Technologies de l'information — Langages de base de données — SQL —

Partie 5: Liants de langage d'hôte (SQL/Liants)

Reference number
ISO/IEC 9075-5:1999(E)

Contents

TABLES

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 9075-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

ISO/IEC 9075 consists of the following parts, under the general title *Information technology — Database languages — SQL*:

— *Part 1: Framework (SQL/Framework)*

— *Part 2: Foundation (SQL/Foundation)*

— *Part 3: Call-Level Interface (SQL/CLI)*

— *Part 4: Persistent Stored Modules (SQL/PSM)*

— *Part 5: Host Language Bindings (SQL/Bindings)*

Annexes A, B, C, D, E, and F of this part of ISO/IEC 9075 are for information only.

Introduction

The organization of this part of ISO/IEC 9075 is as follows:

1) Clause 1, “Scope”, specifies the scope of this part of ISO/IEC 9075.
2) Clause 2, “Normative references”, identifies additional standards that, through reference in this part of ISO/IEC 9075, constitute provisions of this part of ISO/IEC 9075.
3) Clause 3, “Definitions, notations, and conventions”, defines the notations and conventions used in this part of ISO/IEC 9075.
4) Clause 4, “Concepts”, presents concepts used in the definition of Persistent SQL modules.
5) Clause 5, “Lexical elements”, defines the lexical elements of the language.
6) Clause 6, “Scalar expressions”, defines the elements of the language that produce scalar values.
7) Clause 7, “Query expressions”, defines the elements of the language that produce rows and tables of data.
8) Clause 8, “Additional common elements”, defines additional language elements that are used in various parts of the language.
9) Clause 9, “Data assignment rules and routine determination”, specifies the rules for assignments that retrieve data from or store data into SQL-data, and formation rules for set operations.
10) Clause 10, “Schema definition and manipulation”, defines facilities for creating and managing a schema.
11) Clause 11, “SQL-client modules”, defines modules and procedures.
12) Clause 12, “Data manipulation”, defines the data manipulation statements.
13) Clause 13, “Transaction management”, defines the SQL-transaction management statements.
14) Clause 14, “Session management”, defines the SQL-session management statements.
15) Clause 15, “Dynamic SQL”, defines the SQL dynamic statements.
16) Clause 16, “Embedded SQL”, defines the host language embeddings.
17) Clause 17, “Direct invocation of SQL”, defines direct invocation of SQL language.
18) Clause 18, “Diagnostics management”, defines the diagnostics management facilities.
19) Clause 19, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.
20) Clause 20, “Status codes”, defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.
21) Clause 21, “Conformance”, defines the criteria for conformance to this part of ISO/IEC 9075.

22) Annex A, “SQL Conformance Summary”, is an informative Annex. It summarizes the conformance requirements of the SQL language.

23) Annex B, “Implementation-defined elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-defined.

24) Annex C, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-dependent.

25) Annex D, “Deprecated features”, is an informative Annex. It lists features that the responsible Technical Committee intends will not appear in a future revised version of ISO/IEC 9075.

26) Annex E, “Incompatibilities with ISO/IEC 9075:1992”, is an informative Annex. It lists the incompatibilities between this version of ISO/IEC 9075 and ISO/IEC 9075:1992.

27) Annex F, “SQL feature and package taxonomy”, is an informative Annex. It identifies features of the SQL language specified in this part of ISO/IEC 9075 by a numeric identifier and a short descriptive name. This taxonomy is used to specify conformance to Core SQL and may be used to develop other profiles involving the SQL language.

In the text of this part of ISO/IEC 9075, Clauses begin a new odd-numbered page, and in Clause 5, “Lexical elements”, through Clause 21, “Conformance”, Subclauses begin a new page. Any resulting blank space is not significant.

Information technology — Database languages — SQL —

Part 5:
Host Language Bindings (SQL/Bindings)

1 Scope

This part of ISO/IEC 9075 specifies:

— Syntax for embedding SQL-statements in a compilation unit that otherwise conforms to the standard for a particular programming language (host language).

— How an equivalent compilation unit may be derived that conforms to the particular programming language standard. In that equivalent compilation unit, each embedded SQL-statement has been replaced by one or more statements in the host language, some of which invoke an SQL externally-invoked procedure that, when executed, has an effect equivalent to executing the SQL-statement.

— Syntax for direct invocation of SQL-statements.

— SQL language to support dynamic preparation and execution of SQL-statements.

2 Normative references

Insert this paragraph The following standards contain provisions that, through reference in this text, constitute provisions of this part of ISO/IEC 9075. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9075 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 1539-1:1997, *Information technology — Programming languages — Fortran — Part 1: Base language*.

ISO 1989:1985, *Programming languages — COBOL*.
(Endorsement of ANSI X3.23-1985).

ISO 6160:1979, *Programming languages — PL/I*
(Endorsement of ANSI X3.53-1976).

ISO/IEC 7185:1990, *Information technology — Programming languages—Pascal*.

ISO 8652:1995, *Information technology — Programming languages — Ada*.

ISO/IEC 9075-1, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

ISO/IEC 9075-2, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

ISO/IEC 9075-3:1995, *Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)*.

ISO/IEC 9075-4, *Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)*.

ISO/IEC 9899:1990, *Programming languages — C*.

ISO/IEC 9899:1990/Amendment 1:1995, *Amendment to ISO/IEC 9899:1990 — C Integrity*.

ISO/IEC 10206:1991, *Information technology — Programming languages—Extended Pascal*.

ISO/IEC 11756:1992, *Information technology—Programming languages — MUMPS*.

3 Definitions, notations, and conventions

3.1 Definitions

Insert this paragraph For the purposes of this part of ISO/IEC 9075, the definitions given in ISO/IEC 9075-1 and ISO/IEC 9075-2 apply.

3.2 Notation

Insert this paragraph The notation used in this part of ISO/IEC 9075 is defined in ISO/IEC 9075-1.

3.3 Conventions

Insert this paragraph The conventions used in this part of ISO/IEC 9075 is defined in ISO/IEC 9075-1.

3.3.1 Use of terms

3.3.1.1 Other terms

Insert this paragraph An SQL-statement *S1* is said to be executed as a *direct result of executing an SQL-statement* if *S1* is the value of an <SQL statement variable> referenced by an <execute immediate statement> contained in an <externally-invoked procedure> that has been executed, or if *S1* was the value of the <SQL statement variable> that was associated with an <SQL statement name> by a <prepare statement> and that same <SQL statement name> is referenced by an <execute statement> contained in an <externally-invoked procedure> that has been executed.

3.3.2 Relationships to other parts of ISO/IEC 9075

3.3.2.1 Clause, Subclause, and Table relationships

Table 1—Clause, Subclause, and Table relationships

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Clause 1, “Scope”	Clause 1, "Scope"	ISO/IEC 9075-2
Clause 2, “Normative references”	Clause 2, "Normative references"	ISO/IEC 9075-2
Clause 3, “Definitions, notations, and conventions”	Clause 3, "Definitions, notations, and conventions"	ISO/IEC 9075-2

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 3.1, “Definitions”	Subclause 3.1, "Definitions"	ISO/IEC 9075-1
Subclause 3.2, “Notation”	Subclause 6.1, "Notation"	ISO/IEC 9075-1
Subclause 3.3, “Conventions”	Subclause 6.2, "Conventions"	ISO/IEC 9075-1
Subclause 3.3.1, “Use of terms”	Subclause 6.2.3, "Use of terms"	ISO/IEC 9075-1
Subclause 3.3.1.1, “Other terms”	Subclause 6.2.3.7, "Other terms"	ISO/IEC 9075-1
Subclause 3.3.2, “Relationships to other parts of ISO/IEC 9075”	*(none)*	*(none)*
Subclause 3.3.2.1, “Clause, Subclause, and Table relationships”	*(none)*	*(none)*
Subclause 3.4, “Object identifier for Database Language SQL”	Subclause 6.3, "Object identifier for Database Language SQL"	ISO/IEC 9075-1
Clause 4, “Concepts”	Clause 4, "Concepts"	ISO/IEC 9075-2
Subclause 4.1, “Catalogs”	Subclause 4.2.6.1, "Catalogs"	ISO/IEC 9075-1
Subclause 4.2, “SQL-client modules”	Subclause 4.21, "SQL-client modules"	ISO/IEC 9075-2
Subclause 4.3, “SQL-invoked routines”	Subclause 4.23, "SQL-invoked routines"	ISO/IEC 9075-2
Subclause 4.4, “Locators”	Subclause 4.26.4, "Locators"	ISO/IEC 9075-2
Subclause 4.5, “Cursors”	Subclause 4.29, "Cursors"	ISO/IEC 9075-2
Subclause 4.6, “SQL-statements”	Subclause 4.30, "SQL-statements"	ISO/IEC 9075-2
Subclause 4.6.1, “Classes of SQL-statements”	Subclause 4.30.1, "Classes of SQL-statements"	ISO/IEC 9075-2
Subclause 4.6.2, “SQL-statements classified by function”	Subclause 4.30.2, "SQL-statements classified by function"	ISO/IEC 9075-2
Subclause 4.6.3, “SQL-statements and transaction states”	Subclause 4.30.3, "SQL-statements and transaction states"	ISO/IEC 9075-2
Subclause 4.6.4, “Embeddable SQL-statements”	*none*	*none*
Subclause 4.6.5, “Preparable and immediately executable SQL-statements”	*none*	*none*
Subclause 4.6.6, “Directly executable SQL-statements”	*none*	*none*
Subclause 4.7, “Standard programming languages”	Subclause 4.28, "Standard programming languages"	ISO/IEC 9075-2
Subclause 4.8, “Embedded syntax”	*none*	*none*

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 4.9, "SQL dynamic statements"	*none*	*none*
Subclause 4.10, "Direct invocation of SQL"	*none*	*none*
Subclause 4.11, "Privileges and roles"	Subclause 4.31, "Basic security model"	ISO/IEC 9075-2
Subclause 4.12, "SQL-transactions"	Subclause 4.32, "SQL-transactions"	ISO/IEC 9075-2
Subclause 4.13, "SQL-connections"	Subclause 4.33, "SQL-connections"	ISO/IEC 9075-2
Subclause 4.14, "SQL-sessions"	Subclause 4.34, "SQL-sessions"	ISO/IEC 9075-2
Subclause 4.15, "Client-server operation"	Subclause 4.36, "Client-server operation"	ISO/IEC 9075-2
Clause 5, "Lexical elements"	Clause 5, "Lexical elements"	ISO/IEC 9075-2
Subclause 5.1, "<token> and <separator>"	Subclause 5.2, "<token> and <separator>"	ISO/IEC 9075-2
Subclause 5.2, "<literal>"	Subclause 5.3, "<literal>"	ISO/IEC 9075-2
Subclause 5.3, "Names and identifiers"	Subclause 5.4, "Names and identifiers"	ISO/IEC 9075-2
Clause 6, "Scalar expressions"	Clause 6, "Scalar expressions"	ISO/IEC 9075-2
Subclause 6.1, "<value specification> and <target specification>"	Subclause 6.3, "<value specification> and <target specification>"	ISO/IEC 9075-2
Subclause 7.1, "<table reference>"	Subclause 7.6, "<table reference>"	ISO/IEC 9075-2
Subclause 6.2, "<column reference>"	Subclause 6.6, "<column reference>"	ISO/IEC 9075-2
Subclause 6.3, "<interval value expression>"	Subclause 6.29, "<interval value expression>"	ISO/IEC 9075-2
Clause 7, "Query expressions"	Clause 7, "Query expressions"	ISO/IEC 9075-2
Subclause 7.2, "<query specification>"	Subclause 7.11, "<query specification>"	ISO/IEC 9075-2
Clause 8, "Additional common elements"	Clause 10, "Additional common elements"	ISO/IEC 9075-2
Subclause 8.1, "<routine invocation>"	Subclause 10.4, "<routine invocation>"	ISO/IEC 9075-2
Clause 9, "Data assignment rules and routine determination"	Clause 9, "Data assignment rules and routine determination"	ISO/IEC 9075-2
Subclause 9.1, "Retrieval assignment"	Subclause 9.1, "Retrieval assignment"	ISO/IEC 9075-2
Subclause 9.2, "Store assignment"	Subclause 9.2, "Store assignment"	ISO/IEC 9075-2

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	**Corresponding Clause, Subclause, or Table from another part**	**Part containing correspondence**
Clause 10, “Schema definition and manipulation”	Clause 11, "Schema definition and manipulation"	ISO/IEC 9075-2
Subclause 10.1, “<check constraint definition>”	Subclause 11.9, "<check constraint definition>"	ISO/IEC 9075-2
Subclause 10.2, “<view definition>”	Subclause 11.21, "<view definition>"	ISO/IEC 9075-2
Subclause 10.3, “<assertion definition>”	Subclause 11.36, "<assertion definition>"	ISO/IEC 9075-2
Subclause 10.4, “<trigger definition>”	Subclause 11.38, "<trigger definition>"	ISO/IEC 9075-2
Subclause 10.5, “<SQL-invoked routine>”	Subclause 11.49, "<SQL-invoked routine>"	ISO/IEC 9075-2
Clause 11, “SQL-client modules”	Clause 13, "SQL-client modules"	ISO/IEC 9075-2
Subclause 11.1, “<SQL-client module definition>”	Subclause 13.1, "<SQL-client module definition>"	ISO/IEC 9075-2
Subclause 11.2, “Calls to an <externally-invoked procedure>”	Subclause 13.4, "Calls to an <externally-invoked procedure>"	ISO/IEC 9075-2
Subclause 11.3, “<SQL procedure statement>”	Subclause 13.5, "<SQL procedure statement>"	ISO/IEC 9075-2
Clause 12, “Data manipulation”	Clause 14, "Data manipulation"	ISO/IEC 9075-2
Subclause 12.1, “<select statement: single row>”	Subclause 14.5, "<select statement: single row>"	ISO/IEC 9075-2
Subclause 12.2, “<free locator statement>”	Subclause 14.12, "<free locator statement>"	ISO/IEC 9075-2
Clause 13, “Transaction management”	Clause 16, "Transaction management"	ISO/IEC 9075-2
Subclause 13.1, “<commit statement>”	Subclause 16.6, "<commit statement>"	ISO/IEC 9075-2
Clause 14, “Session management”	Clause 18, "Session management"	ISO/IEC 9075-2
Subclause 14.1, “<set catalog statement>”	*none*	*none*
Subclause 14.2, “<set schema statement>”	*none*	*none*
Subclause 14.3, “<set names statement>”	*none*	*none*
Subclause 14.4, “<set path statement>”	*none*	*none*
Subclause 14.5, “<set transform group statement>”	*none*	*none*

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Clause 15, "Dynamic SQL"	*none*	*none*
Subclause 15.1, "Description of SQL descriptor areas"	*none*	*none*
Subclause 15.2, "<allocate descriptor statement>"	*none*	*none*
Subclause 15.3, "<deallocate descriptor statement>"	*none*	*none*
Subclause 15.4, "<get descriptor statement>"	*none*	*none*
Subclause 15.5, "<set descriptor statement>"	*none*	*none*
Subclause 15.6, "<prepare statement>"	*none*	*none*
Subclause 15.7, "<deallocate prepared statement>"	*none*	*none*
Subclause 15.8, "<describe statement>"	*none*	*none*
Subclause 15.9, "<input using clause>"	*none*	*none*
Subclause 15.10, "<output using clause>"	*none*	*none*
Subclause 15.11, "<execute statement>"	*none*	*none*
Subclause 15.12, "<execute immediate statement>"	*none*	*none*
Subclause 15.13, "<dynamic declare cursor>"	*none*	*none*
Subclause 15.14, "<allocate cursor statement>"	*none*	*none*
Subclause 15.15, "<dynamic open statement>"	*none*	*none*
Subclause 15.16, "<dynamic fetch statement>"	*none*	*none*
Subclause 15.17, "<dynamic single row select statement>"	*none*	*none*
Subclause 15.18, "<dynamic close statement>"	*none*	*none*
Subclause 15.19, "<dynamic delete statement: positioned>"	*none*	*none*

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 15.20, "<dynamic update statement: positioned>"	*none*	*none*
Subclause 15.21, "<preparable dynamic delete statement: positioned>"	*none*	*none*
Subclause 15.22, "<preparable dynamic update statement: positioned>"	*none*	*none*
Clause 16, "Embedded SQL"	*none*	*none*
Subclause 16.1, "<embedded SQL host program>"	*none*	*none*
Subclause 16.2, "<embedded exception declaration>"	*none*	*none*
Subclause 16.3, "<embedded SQL Ada program>"	*none*	*none*
Subclause 16.4, "<embedded SQL C program>"	*none*	*none*
Subclause 16.5, "<embedded SQL COBOL program>"	*none*	*none*
Subclause 16.6, "<embedded SQL Fortran program>"	*none*	*none*
Subclause 16.7, "<embedded SQL MUMPS program>"	*none*	*none*
Subclause 16.8, "<embedded SQL Pascal program>"	*none*	*none*
Subclause 16.9, "<embedded SQL PL/I program>"	*none*	*none*
Clause 17, "Direct invocation of SQL"	*none*	*none*
Subclause 17.1, "<direct SQL statement>"	*none*	*none*
Subclause 17.2, "<direct select statement: multiple rows>"	*none*	*none*
Clause 18, "Diagnostics management"	Clause 19, "Diagnostics management"	ISO/IEC 9075-2
Subclause 18.1, "<get diagnostics statement>"	Subclause 19.1, "<get diagnostics statement>"	ISO/IEC 9075-2
Clause 19, "Definition Schema"	Clause 21, "Definition Schema"	ISO/IEC 9075-2
Subclause 19.1, "SQL_LANGUAGES base table"	Subclause 21.37, "SQL_LANGUAGES base table"	ISO/IEC 9075-2

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	**Corresponding Clause, Subclause, or Table from another part**	**Part containing correspondence**
Clause 20, “Status codes”	Clause 22, "Status codes"	ISO/IEC 9075-2
Subclause 20.1, “SQLSTATE”	Subclause 22.1, "SQLSTATE"	ISO/IEC 9075-2
Clause 21, “Conformance”	Clause 23, "Conformance"	ISO/IEC 9075-2
Subclause 21.2, “Claims of conformance”	Subclause 8.1.5, "Claims of conformance"	ISO/IEC 9075-1
Annex A, “SQL Conformance Summary”	Appendix A, "SQL Conformance Summary"	ISO/IEC 9075-2
Annex B, “Implementation-defined elements”	Appendix B, "Implementation-defined elements"	ISO/IEC 9075-2
Annex C, “Implementation-dependent elements”	Appendix C, "Implementation-dependent elements"	ISO/IEC 9075-2
Annex D, “Deprecated features”	Appendix D, "Deprecated features"	ISO/IEC 9075-2
Annex E, “Incompatibilities with ISO/IEC 9075:1992”	Appendix E, "Incompatibilities with ISO/IEC 9075:1992 and ISO/IEC 9075-4:1996"	ISO/IEC 9075-2
Annex F, “SQL feature and package taxonomy”	Appendix F, "SQL feature and package taxonomy"	ISO/IEC 9075-2
Table 1, “Clause, Subclause, and Table relationships”	*none*	*none*
Table 2, “Data types of <key word>s used in the header of SQL descriptor areas”	*none*	*none*
Table 3, “Data types of <key word>s used in SQL item descriptor areas”	*none*	*none*
Table 4, “Codes used for SQL data types in Dynamic SQL”	*none*	*none*
Table 5, “Codes associated with datetime data types in Dynamic SQL”	*none*	*none*
Table 6, “Codes used for <interval qualifier>s in Dynamic SQL”	*none*	*none*
Table 7, “Codes used for input/output SQL parameter modes in Dynamic SQL”	*none*	*none*
Table 8, “<identifier>s for use with <get diagnostics statement>”	Table 25, "<identifier>s for use with <get diagnostics statement>"	ISO/IEC 9075-2
Table 9, “SQL-statement codes”	Table 26, "SQL-statement codes"	ISO/IEC 9075-2
Table 10, “SQLSTATE class and subclass values”	Table 27, "SQLSTATE class and subclass values"	ISO/IEC 9075-2

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Table 12, "SQL/Bindings feature taxonomy and definition for Core SQL"	Table 31, "SQL/Foundation feature taxonomy and definition for Core SQL"	ISO/IEC 9075-2
Table 13, "SQL/Bindings feature taxonomy for features outside Core SQL"	Table 32, "SQL/Foundation feature taxonomy for features outside Core SQL"	ISO/IEC 9075-2

3.4 Object identifier for Database Language SQL

The object identifier for Database Language SQL is defined in ISO/IEC 9075-1 in Subclause 6.3, "Object identifier for Database Language SQL", with the following additions:

Format

```
<Part 5 yes> ::=
      <Part 5 conformance>
      <Part 5 direct> <Part 5 embedded>

<Part 5 conformance> ::=
      5 | sqlbindings199x <left paren> 5 <right paren>

<Part 5 direct> ::=
        <Part 5 direct yes>
      | <Part 5 direct no>

<Part 5 direct yes> ::=
      1 | directyes <left paren> 1 <right paren>

<Part 5 direct no> ::=
      0 | directno <left paren> 0 <right paren>

<Part 5 embedded> ::=
        <Part 5 embedded no>
      | <Part 5 embedded languages>...

<Part 5 embedded no> ::=
      0 | embeddedno <left paren> 0 <right paren>

<Part 5 embedded languages> ::=
        <Part 5 embedded Ada>
      | <Part 5 embedded C>
      | <Part 5 embedded COBOL>
      | <Part 5 embedded Fortran>
      | <Part 5 embedded MUMPS>
      | <Part 5 embedded Pascal>
      | <Part 5 embedded PL/I>

<Part 5 embedded Ada> ::=
      1 | embeddedAda <left paren> 1 <right paren>

<Part 5 embedded C> ::=
      2 | embeddedC <left paren> 2 <right paren>

<Part 5 embedded COBOL> ::=
      3 | embeddedCOBOL <left paren> 3 <right paren>

<Part 5 embedded Fortran> ::=
      4 | embeddedFortran <left paren> 4 <right paren>

<Part 5 embedded MUMPS> ::=
      5 | embeddedMUMPS <left paren> 5 <right paren>

<Part 5 embedded Pascal> ::=
      6 | embeddedPascal <left paren> 6 <right paren>

<Part 5 embedded PL/I> ::=
      7 | embeddedPLI <left paren> 7 <right paren>
```

Syntax Rules

1) Specification of <Part 5 Yes> implies that conformance to this part of ISO/IEC 9075 is claimed.

2) Specification of <Part 5 direct yes> implies that conformance to <direct SQL statement> is claimed.

3) Specification of <Part 5 direct no> implies that conformance to <direct SQL statement> is not claimed.

4) Specification of <Part 5 embedded *language*> implies that conformance to <embedded SQL host program> for the specified *language* is claimed.

5) Specification of <Part 5 embedded no> implies that conformance to <embedded SQL host program> is not claimed.

4 Concepts

4.1 Catalogs

Insert this paragraph The default catalog name for <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s is initially implementation-defined but may be changed by the use of <set catalog statement>s.

4.2 SQL-client modules

Insert this paragraph SQL-session modules are implicitly-created modules for prepared SQL-statements (see Subclause 4.34, "SQL-sessions", in ISO/IEC 9075-2).

Insert this paragraph Each <SQL-client module definition> *M* that is associated with an SQL-session, other than an SQL-session module, may have associated with it a *shadow module M1* that has an implementation-dependent name not equivalent to the <module name> of any other <SQL-client module definition> in the same SQL-session and whose <module authorization clause> specifies “SCHEMA *SN*”, where *SN* is the explicit or implicit <schema name> of the <module authorization clause> of *M*. The <language clause>, the SQL-path, if specified, and the <module character set specification> of *M1* are identical to the corresponding characteristic of *M*. When *M* contains a <module authorization clause> that specifies “FOR STATIC ONLY”, this shadow module effectively contains one <externally-invoked procedure> for each SQL-statement prepared by <prepared statement>s or <execute immediate statement>s contained in *M*.

4.3 SQL-invoked routines

Augment the 6th paragraph The <SQL procedure statement> forming the <routine body> must exclude <SQL dynamic statement>.

4.4 Locators

Augment the 1st paragraph A host variable may be specified to be a *locator* by specifying AS LOCATOR.

Augment the 1st paragraph A host variable specified as a locator may be further specified to be a *holdable locator*.

4.5 Cursors

Insert this paragraph A cursor is also specified by a <dynamic declare cursor> or an <allocate cursor statement>. A cursor specified by a <dynamic declare cursor> is a *declared dynamic cursor*. A cursor specified by an <allocate cursor statement> is an *extended dynamic cursor*. A *dynamic cursor* is either a declared dynamic cursor or an extended dynamic cursor.

Insert this paragraph For every <dynamic declare cursor> in an <SQL-client module definition>, a cursor is effectively created when an SQL-transaction (see Subclause 4.32, "SQL-transactions", in ISO/IEC 9075-2) referencing the <SQL-client module definition> is initiated. An extended dynamic cursor is also effectively created when an <allocate cursor statement> is executed within an SQL-session and destroyed when that SQL-session is terminated.

Insert this paragraph In addition, a dynamic cursor is destroyed when a <deallocate prepared statement> is executed that deallocates the prepared statement on which the dynamic cursor is based.

Insert this paragraph A cursor is also placed in the open state by a <dynamic open statement> and returned to the closed state by a <dynamic close statement>.

Insert this paragraph A <dynamic fetch statement> positions an open cursor on a specified row of the cursor's ordering and retrieves the values of the columns of that row. A <dynamic update statement: positioned> updates the current row of the cursor. A <dynamic delete statement: positioned> deletes the current row of the cursor.

4.6 SQL-statements

4.6.1 Classes of SQL-statements

Insert this paragraph Most SQL-statements can be prepared for execution and executed in additional ways. These are:

— In an embedded SQL host program, in which case they are prepared when the embedded SQL host program is preprocessed (see Subclause 4.8, "Embedded syntax").

— Being prepared and executed by the use of SQL-dynamic statements (which are themselves executed in an SQL-client module or an embedded SQL host program—see Subclause 4.9, "SQL dynamic statements").

— Direct invocation, in which case they are effectively prepared immediately prior to execution (see Subclause 4.10, "Direct invocation of SQL").

Insert this paragraph There are at least three additional ways of classifying SQL-statements:

— According to whether or not they may be embedded.

— According to whether they may be dynamically prepared and executed.

— According to whether or not they may be directly executed.

4.6.2 SQL-statements classified by function

Insert this paragraph The following are other main classes of SQL-statements:

— SQL-dynamic statements

— SQL embedded exception declaration

Insert this paragraph The following are additional SQL-data statements:

— <dynamic declare cursor>

— <allocate cursor statement>

— <dynamic select statement>

— <dynamic open statement>

— <dynamic close statement>

— <dynamic fetch statement>

— <direct select statement: multiple rows>

— <dynamic single row select statement>

Insert this paragraph The following are additional SQL-data change statements:

— <dynamic delete statement: positioned>

— <preparable dynamic delete statement: positioned>

— <dynamic update statement: positioned>

— <preparable dynamic update statement: positioned>

Insert this paragraph The following are additional SQL-session statements:

— <set catalog statement>

— <set schema statement>

— <set names statement>

— <set path statement>

— <set transform group statement>

Insert this paragraph The following are the SQL-dynamic statements:

— <execute immediate statement>

— <allocate descriptor statement>

— <deallocate descriptor statement>

— <get descriptor statement>

— <set descriptor statement>

— <prepare statement>

— <deallocate prepared statement>

— <describe input statement>

— <describe output statement>

— <execute statement>

Insert this paragraph The following is the SQL embedded exception declaration:

— <embedded exception declaration>

4.6.3 SQL-statements and transaction states

Insert this paragraph Whether or not an <execute immediate statement> starts a transaction depends on the content of the <SQL statement variable> referenced by the <execute immediate statement> at the time it is executed. Whether or not an <execute statement> starts a transaction depends on the content of the <SQL statement variable> referenced by the <prepare statement> at the time the prepared statement referenced by the <execute statement> was prepared. In both cases, if the content of the <SQL statement variable> was a transaction-initiating SQL-statement, then the <execute immediate statement> or <execute statement> is treated as a transaction-initiating statement; otherwise it is not treated as a transaction-initiating statement.

Insert this paragraph The following additional SQL-statements are transaction-initiating SQL-statements, *i.e.*, if there is no current transaction, and a statement of this class is executed, a transaction is initiated:

— <allocate cursor statement>

— <dynamic open statement>

— <dynamic close statement>

— <dynamic fetch statement>

— <direct select statement: multiple rows>

— <dynamic single row select statement>

— <dynamic delete statement: positioned>

— <preparable dynamic delete statement: positioned>

— <dynamic update statement: positioned>

— <preparable dynamic update statement: positioned>

— The following SQL-dynamic statements:

  • <describe input statement>
  • <describe output statement>
  • <allocate descriptor statement>
  • <deallocate descriptor statement>
  • <get descriptor statement>
  • <set descriptor statement>
  • <prepare statement>
  • <deallocate prepared statement>

Insert this paragraph The following additional SQL-statements are not transaction-initiating SQL-statements, *i.e.*, if there is no current transaction, and a statement of this class is executed, no transaction is initiated.

— SQL embedded exception declarations

— The following SQL-data statements:

- <dynamic declare cursor>
- <dynamic select statement>

4.6.4 Embeddable SQL-statements

The following SQL-statements are embeddable in an embedded SQL host program, and may be the <SQL procedure statement> in an <externally-invoked procedure> in an <SQL-client module definition>:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— All SQL-dynamic statements

— All SQL-diagnostics statements

— The following SQL-data statements:

- <allocate cursor statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>
- <select statement: single row>
- <insert statement>
- <delete statement: searched>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <update statement: searched>

- <update statement: positioned>
- <dynamic update statement: positioned>
- <free locator statement>

— The following SQL-control statements:

- <call statement>

The following SQL-statements are embeddable in an embedded SQL host program, and may occur in an <SQL-client module definition>, though not in an <externally-invoked procedure>:

— <temporary table declaration>

— <declare cursor>

— <dynamic declare cursor>

The following SQL-statements are embeddable in an embedded SQL host program, but may not occur in an <SQL-client module definition>:

— SQL embedded exception declarations

Consequently, the following SQL-data statements are not embeddable in an embedded SQL host program, nor may they occur in an <SQL-client module definition>, nor be the <SQL procedure statement> in an <externally-invoked procedure> in an <SQL-client module definition>:

— <dynamic select statement>

— <dynamic single row select statement>

— <direct select statement: multiple rows>

— <preparable dynamic delete statement: positioned>

— <preparable dynamic update statement: positioned>

4.6.5 Preparable and immediately executable SQL-statements

The following SQL-statements are preparable:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-session statements

— The following SQL-data statements:

- <delete statement: searched>
- <dynamic select statement>
- <dynamic single row select statement>
- <insert statement>

  - <update statement: searched>
  - <preparable dynamic delete statement: positioned>
  - <preparable dynamic update statement: positioned>
  - <preparable implementation-defined statement>
  - <free locator statement>

— The following SQL-control statements:

  - <call statement>

Consequently, the following SQL-statements are not preparable:

— All SQL-connection statements

— All SQL-dynamic statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

  - <allocate cursor statement>
  - <open statement>
  - <dynamic open statement>
  - <close statement>
  - <dynamic close statement>
  - <fetch statement>
  - <dynamic fetch statement>
  - <select statement: single row>
  - <delete statement: positioned>
  - <dynamic delete statement: positioned>
  - <update statement: positioned>
  - <dynamic update statement: positioned>
  - <direct select statement: multiple rows>
  - <temporary table declaration>
  - <declare cursor>
  - <dynamic declare cursor>

Any preparable SQL-statement can be executed immediately, with the exception of:

— <dynamic select statement>

— <dynamic single row select statement>

4.6.6 Directly executable SQL-statements

The following SQL-statements may be executed directly:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— The following SQL-data statements:

- <temporary table declaration>
- <direct select statement: multiple rows>
- <insert statement>
- <delete statement: searched>
- <update statement: searched>

Consequently, the following SQL-statements may not be executed directly:

— All SQL-dynamic statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

- <declare cursor>
- <dynamic declare cursor>
- <allocate cursor statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>

- <select statement: single row>
- <dynamic select statement>
- <dynamic single row select statement>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

— <free locator statement>

— <hold locator statement>

4.7 Standard programming languages

Insert this paragraph This part of ISO/IEC 9075 specifies a mechanism whereby SQL language may be embedded in programs that otherwise conform to any of the same specified programming language standards.

NOTE 1 – In this part of ISO/IEC 9075, for the purposes of interfacing with programming languages, the data types DATE, TIME, TIMESTAMP, and INTERVAL must be converted to or from character strings in those programming languages by means of a <cast specification>. It is anticipated that future evolution of programming language standards will support data types corresponding to these four SQL data types; this standard will then be amended to reflect the availability of those corresponding data types.

The data types CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT are also mapped to character strings in the programming languages. However, because the facilities available in the programming languages do not provide the same capabilities as those available in SQL, there must be agreement between the host program and SQL regarding the specific format of the character data being exchanged. Specific syntax for this agreement is provided in this part of ISO/IEC 9075. For standard programming languages C, COBOL, Fortran, and Pascal, bit strings are mapped to character variables in the host language in a manner described in Subclause 16.1, “<embedded SQL host program>”. For standard programming languages Ada and PL/I, bit string variables are directly supported.

For standard programming languages C and COBOL, BOOLEAN values are mapped to integer variables in the host language. For standard programming languages Ada, Fortran, Pascal, and PL/I, BOOLEAN variables are directly supported.

For the purposes of interfacing with programming languages, the data type ARRAY must be converted to a locator (see Subclause 4.26.4, "Locators", in ISO/IEC 9075-2).

For the purposes of interfacing with programming languages, user-defined types must be handled with a locator (see Subclause 4.26.4, "Locators", in ISO/IEC 9075-2) or transformed to another SQL data type that has a defined mapping to the host language (see Subclause 4.8.5, "Transforms for user-defined types", in ISO/IEC 9075-2).

4.8 Embedded syntax

An <embedded SQL host program> (<embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL COBOL program>, <embedded SQL Fortran program>, <embedded SQL MUMPS program>, <embedded SQL Pascal program>, or <embedded SQL PL/I program>) is a compilation unit that consists of programming language text and SQL text. The programming language text shall conform to the requirements of a specific standard programming language. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s, as defined in this International Standard. This allows database applications to be expressed in a hybrid form in which SQL-statements are embedded directly in a compilation unit. Such a hybrid compilation unit is defined to be equivalent to a standard compilation unit in which the SQL-statements have been replaced by standard procedure or subroutine calls of <externally-invoked procedure>s in a separate SQL-client module, and in which each has been removed and the declarations contained therein have been suitably transformed into standard host-language syntax.

If an <embedded SQL host program> contains an <embedded authorization declaration>, then it shall be the first statement or declaration in the <embedded SQL host program>. The <embedded authorization declaration> is not replaced by a procedure or subroutine call of an <externally-invoked procedure>, but is removed and replaced by syntax associated with the <SQL-client module definition>'s <module authorization clause>.

An implementation may reserve a portion of the name space in the <embedded SQL host program> for the names of procedures or subroutines that are generated to replace SQL-statements and for program variables and branch labels that may be generated as required to support the calling of these procedures or subroutines; whether this reservation is made is implementation-defined. They may similarly reserve name space for the <module name> and <procedure name>s of the generated <SQL-client module definition> that may be associated with the resulting standard compilation unit. The portion of the name space to be so reserved, if any, is implementation-defined.

4.9 SQL dynamic statements

In many cases, the SQL-statement to be executed can be coded into an <SQL-client module definition> or into a compilation unit using the embedded syntax. In other cases, the SQL-statement is not known when the program is written and will be generated during program execution. An <execute immediate statement> can be used for a one-time preparation and execution of an SQL-statement. A <prepare statement> is used to prepare the generated SQL-statement for subsequent execution. A <deallocate prepared statement> is used to deallocate SQL-statements that have been prepared with a <prepare statement>. A description of the input dynamic parameters for a prepared statement can be obtained by execution of a <describe input statement>. A description of the resultant columns of a <dynamic select statement> or <dynamic single row select statement> can be obtained by execution of a <describe output statement>. A description of the output dynamic parameters of a statement that is neither a <dynamic select statement> nor a <dynamic single row select statement> can be obtained by execution of a <describe output statement>. For a statement other than a <dynamic select statement>, an <execute statement> is used to associate parameters with the prepared statement and execute it as though it had been coded when the program was written. For a <dynamic select statement>, the prepared <cursor specification> is associated with a cursor via a <dynamic declare cursor> or <allocate cursor statement>. The cursor can be opened and dynamic parameters can be associated with the cursor with a <dynamic open statement>. A <dynamic fetch statement> positions an open cursor on a specified row and retrieves the values of the columns of that row. A <dynamic close statement> closes a cursor that was opened with a <dynamic open statement>. A <dynamic delete statement: positioned> is used to delete rows through a dynamic cursor. A <dynamic update statement: positioned> is used to update rows through a dynamic cursor. A <preparable dynamic delete statement: positioned> is used to delete rows through a dynamic cursor when the precise format of the statement isn't known until runtime. A <preparable dynamic update statement: positioned> is used to update rows through a dynamic cursor when the precise format of the statement isn't known until runtime.

The interface for input dynamic parameters and output dynamic parameters for a prepared statement and for the resulting values from a <dynamic fetch statement> or the execution of a prepared <dynamic single row select statement> can be either a list of dynamic parameters or embedded variables or an SQL descriptor area. An SQL descriptor area consists of one or more item descriptor areas, together with a header that includes a count of the number of those item descriptor areas. The header of an SQL descriptor area consists of the fields in Table 2, "Data types of <key word>s used in the header of SQL descriptor areas", in Subclause 15.1, "Description of SQL descriptor areas". Each item descriptor area consists of the fields specified in Table 3, "Data types of <key word>s used in SQL item descriptor areas", in Subclause 15.1, "Description of SQL descriptor areas". The SQL descriptor area is allocated and maintained by the system with the following statements: <allocate descriptor statement>, <deallocate descriptor statement>, <set descriptor statement>, and <get descriptor statement>.

An SQL descriptor area is identified by a <descriptor name>, which is a <simple value specification> whose value is an <identifier>. Two <descriptor name>s identify the same SQL descriptor area if their values, with leading and trailing <space>s removed, are equivalent according to the rules for <identifier> comparisons in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

Dynamic statements can be identified by <statement name>s or by <extended statement name>s. Similarly, dynamic cursors can be identified by <cursor name>s and by <extended cursor name>s. The non-extended names are <identifier>s. The extended names are <target specification>s whose values are <identifier>s used to identify the statement or cursor. Two extended names are equivalent if their values, with leading and trailing <space>s removed, are equivalent according to the rules for <identifier> comparison in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

An SQL descriptor area name may be defined as global or local. Similarly, an extended statement name or extended cursor name may be global or local. The scope of a global name is the SQL-session. The scope of a local name is the <SQL-client module definition> in which it appears. A reference to an entity in which one specifies a global scope is valid only if the entity was defined as global and if the reference is from the same SQL-session in which it was defined. A reference to an entity in which one specifies a local scope is valid only if the entity was defined as local and if the reference is from the same <SQL-client module definition> in which it was defined. (The scope of non-extended statement names and non-extended cursor names is always local.)

NOTE 2 – The namespace of non-extended statement names and non-extended cursor names is different from the namespace of extended statement names that are specified to be LOCAL and extended cursor names that are specified to be LOCAL, respectively.

The relationships between prepared statements and <SQL-client module definition>s are:

— Within an SQL-session, all global prepared statements (prepared statements with global statement names) belong to the SQL-session module.

— Within an SQL-session, each local prepared statement (a prepared statement with a local statement name) prepared in an <SQL-client module definition> containing a <module authorization clause> that does not specify "AUTHORIZATION <module authorization identifier>" belongs to the <SQL-client module definition> that contains the <prepare statement> or <execute immediate statement> with which it is prepared.

— Within an SQL-session, each local prepared statement (a prepared statement with a local statement names) prepared in an <SQL-client module definition> containing a <module authorization clause> that contains "AUTHORIZATION <module authorization identifier>" and does not contain "FOR STATIC ONLY" belongs to the <SQL-client module definition> that contains the <prepare statement> or <execute immediate statement> with which it is prepared.

— Within an SQL-session, each local prepared statement (a prepared statement with a local statement names) prepared in an <SQL-client module definition> *M* containing a <module authorization clause> that contains "AUTHORIZATION <module authorization identifier> FOR STATIC ONLY" belongs to a *shadow module* that has an implementation-dependent name not equivalent to the <module name> of any other <SQL-client module definition> in the same SQL-session and whose <module authorization clause> specifies "SCHEMA *SN*", where *SN* is the explicit or implicit <schema name> of the <module authorization clause> of *M*.

NOTE 3 – The SQL-session module is defined in Subclause 4.34, "SQL-sessions", in ISO/IEC 9075-2.

Dynamic execution of SQL-statements can generally be accomplished in two different ways. Statements can be *prepared* for execution and then later executed one or more times; when the statement is no longer needed for execution, it can be *released* by the use of a <deallocate prepared statement>. Alternatively, a statement that is needed only once can be executed without the preparation step—it can be *executed immediately* (not all SQL-statements can be executed immediately).

Many SQL-statements can be written to use "parameters" (which are manifested in static execution of SQL-statements as host parameters in <SQL procedure statement>s contained in <externally-invoked procedure>s in <SQL-client module definition>s or as host variables in <embedded SQL statement>s contained in <embedded SQL host program>s). In SQL-statements that are executed dynamically, the parameters are called dynamic parameters (<dynamic parameter specification>s) and are represented in SQL language by a <question mark> (?).

In many situations, an application that generates an SQL-statement for dynamic execution knows in detail the required characteristics (*e.g.*, <data type>, <length>, <precision>, <scale>, *etc.*) of each of the dynamic parameters used in the statement; similarly, the application may also know in detail the characteristics of the values that will be returned by execution of the statement. However, in other cases, the application may not know this information to the required level of detail; it is possible in some cases for the application to ascertain the information from the Information Schema, but in other cases (*e.g.*, when a returned value is derived from a computation instead of simply from a column in a table, or when dynamic parameters are supplied) this information is not generally available except in the context of preparing the statement for execution.

To provide the necessary information to applications, SQL permits an application to request the SQL-server to *describe* a prepared statement. The description of a statement identifies the number of input dynamic parameters (*describe input*) and their data type information or it identifies the number of output dynamic parameters or values to be returned (*describe output*) and their data type information. The description of a statement is placed into the SQL descriptor areas already mentioned.

Many, but not all, SQL-statements can be prepared and executed dynamically.

NOTE 4 – The complete list of statements that may be dynamically prepared and executed is defined in Subclause 4.6.5, "Preparable and immediately executable SQL-statements".

Certain "set statements" (<set catalog statement>, <set schema statement>, <set names statement>, and <set path statement>) have no effect other than to set up default information (catalog name, schema name, character set, and SQL path, respectively) to be applied to other SQL-statements that are prepared or executed immediately or that are invoked directly.

Syntax errors and Access Rule violations caused by the preparation or immediate execution of <preparable statement>s are identified when the statement is prepared (by <prepare statement>) or when it is executed (by <execute statement> or <execute immediate statement>); executed (by <execute statement> or <execute immediate statement>); such violations are indicated by the raising of an exception condition.

4.10 Direct invocation of SQL

Direct invocation of SQL is a mechanism for executing direct SQL-statements, known as <direct SQL statement>s. In direct invocation of SQL, the method of invoking <direct SQL statement>s, the method of raising conditions that result from the execution of <direct SQL statement>s, the method of accessing the diagnostics information that results from the execution of <direct SQL statement>s, and the method of returning the results are implementation-defined.

4.11 Privileges and roles

Insert this paragraph For direct SQL, the SQL-session user identifier is always the current authorization identifier.

4.12 SQL-transactions

Insert this paragraph The operations comprising an SQL-transaction may also be performed by the direct invocation of SQL.

Insert this paragraph It is implementation-defined whether or not the dynamic execution of an <SQL dynamic data statement> is permitted to occur within the same SQL-transaction as the dynamic execution of an SQL-schema statement. If it does occur, then the effect on any open cursor, prepared dynamic statement, or deferred constraint is implementation-defined. There may be additional implementation-defined restrictions, requirements, and conditions. If any such restrictions, requirements, or conditions are violated, then an implementation-defined exception condition or a completion condition *warning* with an implementation-defined subclass code is raised.

Insert this paragraph Each direct invocation of SQL that executes an SQL-statement of an SQL-transaction is associated with that SQL-transaction. An SQL-transaction is initiated when no SQL-transaction is currently active by direct invocation of SQL that results in the execution of a transaction-initiating <direct SQL statement>.

4.13 SQL-connections

Insert this paragraph An SQL-connection may also be terminated by the last execution of a <direct SQL statement> through the direct invocation of SQL. The mechanism and rules by which an SQL-implementation determines whether a direct invocation of SQL is the last execution of a <direct SQL statement> are implementation-defined.

4.14 SQL-sessions

Replace 3rd paragraph Within an SQL-session, module local temporary tables are effectively created by <temporary table declaration>s. Module local temporary tables are accessible only to invocations of <externally-invoked procedure>s in the SQL-client module or SQL-session module in which they are created. The definitions of module local temporary tables persist until the end of the SQL-session.

Insert this paragraph An *SQL-session* also spans the execution of a sequence of consecutive SQL-statements invoked by the direct invocation of SQL.

Insert this paragraph An SQL-session has an SQL-session module that is different from any other <SQL-client module definition> that exists simultaneously in the SQL-environment. The SQL-session module contains the global prepared SQL-statements that belong to the SQL-session. The SQL-session module contains a <module authorization clause> that specifies SCHEMA <schema name>, where the value of <schema name> is implementation-dependent.

Insert this paragraph An SQL-session has a default catalog name that is used to effectively qualify unqualified <schema name>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default catalog name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set catalog statement> or <set schema statement>.

Insert this paragraph An SQL-session has a default unqualified schema name that is used to effectively qualify unqualified <schema qualified name>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default unqualified schema name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set schema statement>.

Insert this paragraph An SQL-session has a SQL-path that is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The SQL-path is initially set to an implementation-defined value, but can subsequently be changed by the successful execution of a <set path statement>.

Insert this paragraph The text defining the SQL-path can be referenced by using the <general value specification> CURRENT_PATH.

Insert this paragraph An SQL-session has a default transform group name and one or more user-defined type name-transform group name pairs that are used to identify the group of transform functions for every user-defined type that is referenced in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The transform group name for a given user-defined type name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set transform group statement>.

Insert this paragraph The text defining the transform group names associated with the SQL-session can be referenced using two mechanisms: the <general value specification> "CURRENT_TRANSFORM_GROUP_FOR_TYPE <user-defined type>", which evaluates to the name of the transform group associated with the specified data type, and the <general value specification> "CURRENT_DEFAULT_TRANSFORM_GROUP", which evaluates to the name of the transform group associated with all types that have no type-specific transform group specified for them.

Insert this paragraph An SQL-session has a default character set name that is used to identify the character set in which <preparable statement>s are represented when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement>. The default character set name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set names statement>.

Augment the 4th paragraph Within an SQL-session, locators are effectively created whenever a host variable that is specified as a binary large object locator, character large object locator, array locator, or user-defined type locator is assigned a value of binary large object type, character large object type, array type, or user-defined type, respectively. A host variable that is a locator may be *holdable* or *nonholdable*.

Insert this paragraph The SQL-session context also comprises:

— The SQL-session module.

— The current default catalog name.

— The current default unqualified schema name.

— The current character set name substitution value.

— The text defining the SQL-path.

— The contents of all SQL dynamic descriptor areas.

— The text defining the default transform group name.

— The text defining the user-defined type name-transform group name pair for each user-defined type explicitly set by the user.

4.15 Client-server operation

Insert this paragraph <direct SQL statement>s containing an <SQL connection statement> or an <SQL diagnostics statement> are processed by the SQL-client. Other <direct SQL statement>s are processed by the SQL-server.

5 Lexical elements

5.1 <token> and <separator>

Function

Specify lexical units (tokens and separators) that participate in SQL language.

Format

```
<delimiter token> ::=
       !! All alternatives from ISO/IEC 9075-2
       | <double period>

<double period> ::= ..

<non-reserved word> ::=
         !! All alternatives from ISO/IEC 9075-2

       | DEGREE | DYNAMIC_FUNCTION | DYNAMIC_FUNCTION_CODE

       | RETURNED_CARDINALITY

       | STRUCTURE

       | TOP_LEVEL_COUNT

<reserved word> ::=
         !! All alternatives from ISO/IEC 9075-2

       | CURRENT_DEFAULT_TRANSFORM_GROUP
       | CURRENT_TRANSFORM_GROUP_FOR_TYPE

       | DYNAMIC

       | NESTING
```

Syntax Rules

None.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.2 <literal>

Function

Specify a non-null value.

Format

No additional Format items.

Syntax Rules

1) Augments SR8) Case:

   a) If a <character set specification> is not specified in a <character string literal>, then the set of characters contained in the <character string literal> shall be wholly contained in the character set of the SQL-client module or SQL-session module that contains the <character string literal>.

   b) Otherwise, there shall be no <separator> between the <introducer> and the <character set specification>, and the set of characters contained in the <character string literal> shall be wholly contained in the character set specified by the <character set specification>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.3 Names and identifiers

Function

Specify names.

Format

```
<SQL statement name> ::=
          <statement name>
        | <extended statement name>

<statement name> ::= <identifier>

<extended statement name> ::=
        [ <scope option> ] <simple value specification>

<dynamic cursor name> ::=
          <cursor name>
        | <extended cursor name>

<extended cursor name> ::=
        [ <scope option> ] <simple value specification>

<descriptor name> ::=
        [ <scope option> ] <simple value specification>

<scope option> ::=
          GLOBAL
        | LOCAL
```

Syntax Rules

1) Insert before SR4)a) If the <local or schema qualified name> is contained, without an intervening <schema definition>, in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default <unqualified schema name> for the SQL-session is implicit.

2) Insert before SR11)a) If the <schema qualified name> is contained, without an intervening <schema definition>, in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default <unqualified schema name> for the SQL-session is implicit.

3) Insert before SR12)a) If the <unqualified schema name> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default catalog name for the SQL-session is implicit.

4) Insert this SR The <simple value specification> of <extended statement name> or <extended cursor name> shall not be a <literal>.

5) Insert this SR The declared type of the <simple value specification> of <extended statement name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

6) Insert this SR The declared type of the <simple value specification> of <extended cursor name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

7) Insert this SR The declared type of the <simple value specification> of <descriptor name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

8) Insert this SR In a <descriptor name>, <extended statement name>, or <extended cursor name>, if a <scope option> is not specified, then a <scope option> of LOCAL is implicit.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR The value of an <extended statement name> identifies a statement prepared by the execution of a <prepare statement>. If a <scope option> of GLOBAL is specified, then the scope of the <extended statement name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the <extended statement name> is further restricted to the <SQL-client module definition> in which the <extended statement name> appears.

2) Insert this GR A <dynamic cursor name> identifies a cursor in an <SQL dynamic statement>.

3) Insert this GR A <descriptor name> identifies a descriptor area.

4) Insert this GR A <statement name> identifies a statement prepared by the execution of a <prepare statement>. The scope of a <statement name> is the <SQL-client module definition> in which it appears and the current SQL-session.

5) Insert this GR The value of an <extended cursor name> identifies a cursor created by the execution of an <allocate cursor statement>. If a <scope option> of GLOBAL is specified, then the scope of the <extended cursor name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the <extended cursor name> is further restricted to the <SQL-client module definition> in which the <extended cursor name> appears.

6) Insert this GR A <descriptor name> identifies an SQL descriptor area created by the execution of an <allocate descriptor statement>. If a <scope option> of GLOBAL is specified, then the scope of the <descriptor name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the <descriptor name> is further restricted to the <SQL-client module definition> in which the <descriptor name> appears.

Conformance Rules

1) Insert this CR Without Feature B032, “Extended dynamic SQL”, conforming SQL language shall not contain any <extended statement name> or <extended cursor name>.

2) Insert this CR Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <SQL statement name>, <dynamic cursor name>, or <descriptor name>.

6 Scalar expressions

6.1 <value specification> and <target specification>

Function

Specify one or more values, host parameters, SQL parameters, dynamic parameters, or host variables.

Format

```
<general value specification> ::=
          !! All alternatives from ISO/IEC 9075-2
        | <dynamic parameter specification>
        | <embedded variable specification>
        | CURRENT_DEFAULT_TRANSFORM_GROUP
        | CURRENT_TRANSFORM_GROUP_FOR_TYPE <user-defined type>

<simple value specification> ::=
          !! All alternatives from ISO/IEC 9075-2
        | <embedded variable name>

<target specification> ::=
          !! All alternatives from ISO/IEC 9075-2
        | <dynamic parameter specification>
        | <embedded variable specification>

<simple target specification> ::=
          !! All alternatives from ISO/IEC 9075-2
        | <embedded variable name>

<dynamic parameter specification> ::= <question mark>

<embedded variable specification> ::=
        <embedded variable name> [ <indicator variable> ]

<indicator variable> ::=
        [ INDICATOR ] <embedded variable name>
```

Syntax Rules

1) Insert this SR The declared type of an <indicator variable> shall be exact numeric with a scale of 0 (zero).

2) Insert this SR Each <embedded variable name> shall be contained in an <embedded SQL statement>.

3) Insert this SR Each <dynamic parameter specification> shall be contained in a <preparable statement> that is dynamically prepared in the current SQL-session through the execution of a <prepare statement>.

4) Insert this SR The data type of CURRENT_DEFAULT_TRANSFORM_GROUP and of CURRENT_TRANSFORM_GROUP_FOR_TYPE <user-defined type> is a character string. Whether the character string is fixed length or variable length, and its length if fixed length or maximum length if variable length, are implementation-defined. The character set of the character string is SQL_IDENTIFIER.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR A <dynamic parameter specification> identifies a parameter used by a dynamically prepared statement.

2) Insert this GR An <embedded variable specification> identifies a host variable or a host variable and an indicator variable.

3) Augments GR3) A <target specification> specifies a parameter used in a dynamically prepared statement or a host variable that can be assigned a value.

4) Insert this GR If an <embedded variable specification> contains an <indicator variable> and the value of the indicator variable is negative, then the value specified by the <embedded variable specification> is null; otherwise, the value specified by a <embedded variable specification> is the value of the host variable identified by the <embedded variable name>.

5) Insert this GR The value specified by CURRENT_DEFAULT_TRANSFORM_GROUP is the character string that represents the default transform group name associated with the SQL-session.

6) Insert this GR The value specified by CURRENT_TRANSFORM_GROUP_FOR_TYPE <user-defined type> is the character string that represents the transform group name associated with the data type specified by <user-defined type>.

Conformance Rules

1) Insert this CR Without Feature F611, “Indicator data types”, the specific declared types of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

2) Insert this CR Without Feature B031, “Basic dynamic SQL”, a <general value specification> shall not be a <dynamic parameter specification>.

6.2 <column reference>

Function

Reference a column.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

1) Insert this AR If *CR* is a <column reference> whose qualifying table is a base table or a viewed derived table and that is contained in any of:

   — A <query expression> simply contained in a <direct select statement: multiple rows>.

   — A <sort specification list> contained in a <direct select statement: multiple rows>.

   — A <query specification> contained in a <dynamic single row select statement>.

   then let *C* be the column referenced by *CR*.

   Case:

   a) If <column reference> is contained in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include SELECT for *C*.

   b) Otherwise, the current privileges shall include SELECT for *C*.

   NOTE 5 – “User privileges” and “applicable privileges” are defined in Subclause 10.5, "<privileges>", in ISO/IEC 9075-2.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6.3 <interval value expression>

Function

Specify an interval value.

Format

```
<interval primary> ::=
         <value expression primary> [ <interval qualifier> ]
       | <interval value function>
```

Syntax Rules

1) Insert this SR An <interval primary> shall specify <interval qualifier> only if the <interval primary> specifies a <dynamic parameter specification>.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR3)a) If *IP* immediately contains a <value expression primary> *VEP* and an explicit <interval qualifier> *IQ*, then the value of *IP* is computed by:

   CAST (*VEP* AS INTERVAL *IQ*)

Conformance Rules

No additional Conformance Rules.

7 Query expressions

7.1 <table reference>

Function

Reference a table.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

1) Insert this AR If the <table reference> is contained in a <direct select statement: multiple rows> then the current privileges shall include SELECT for at least one column of *T*.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

>

7.2 <query specification>

Function

Specify a table derived from the result of a <table expression>.

Format

No additional Format items.

Syntax Rules

1) Insert after SR16)a)ii) An <indicator variable>.

2) Insert after SR16)a)ii) A <dynamic parameter specification>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

8 Additional common elements

8.1 <routine invocation>

Function

Invoke an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Replace SR8)c)i)4)A) If A_i is an <embedded variable name> or a <host parameter specification>, then P_i shall be assignable to A_i, according to the Syntax Rules of Subclause 9.1, "Retrieval assignment", with A_i and P_i as *TARGET* and *VALUE*, respectively.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9 Data assignment rules and routine determination

9.1 Retrieval assignment

Function

Specify rules for assignments to targets that do not support null values or that support null values with indicator parameters (*e.g.*, assigning SQL-data to host variables).

Syntax Rules

1) Replace SR3)a) If *T* is either a locator parameter of an external routine, a locator variable, or a host parameter that is a character large object locator parameter, then the declared type of *V* shall be character large object type and the character string type of *T* and the declared type of *V* shall be mutually assignable.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR3) If *V* is the null value and *T* is a host variable, then

   Case:

   a) If an indicator variable is specified for *T*, then that indicator variable is set to −1.

   b) If no indicator variable is specified for *T*, then an exception condition is raised: *data exception — null value, no indicator parameter*.

2) Insert after GR3) If *V* is not the null value, *T* is a host variable, and *T* has an indicator variable, then

   Case:

   a) If the declared type of *T* is character string, bit string, or binary string and the length in characters, bits, or octets, respectively, *M* of *V* is greater than the length in characters, bits, or octets, respectively, of *T*, then the indicator parameter is set to *M*. If *M* exceeds the maximum value that the indicator parameter can contain, then an exception condition is raised: *data exception — indicator overflow*.

   b) Otherwise, the indicator variable is set to 0 (zero).

9.2 Store assignment

Function

Specify rules for assignments where the target permits null without the use of indicator variables, such as storing SQL-data.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR2)a)iii) If *V* is a host variable and contains an indicator variable, then

   Case:

   1) If the value of the indicator variable is equal to −1, then *T* is set to the null value.

   2) If the value of the indicator variable is less than −1, then an exception condition is raised: *data exception — invalid indicator parameter value.*

9.3 Data types of results of aggregations

Function

Specify the result data type of the result of an aggregation over values of compatible data types, such as <case expression>s, <collection value expression>s, or a column in the result of a <query expression>.

Syntax Rules

1) Replace SR1) Let *IDTS* be a set of data types specified in an application of this Subclause. let *DTS* be the set of data types in *IDTS* excluding any data types that are undefined. If the cardinality of *DTS* is 0 (zero), then the result data type is undefined and no further Rules of this Subclause are evaluated.

   NOTE 6 –

   2) The notion of "undefined data type" is defined in Subclause 15.6, "<prepare statement>".

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

10 Schema definition and manipulation

10.1 <check constraint definition>

Function

Specify a condition for the SQL-data.

Format

No additional Format items.

Syntax Rules

1) Insert this SR The <search condition> shall also not contain an <embedded variable specification> or a <dynamic parameter specification>.

Access Rules

1) No additional Access Rules.

General Rules

1) No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.2 <view definition>

Function

Define a viewed table.

Format

No additional Format items.

Syntax Rules

1) Insert this SR The <view definition> shall not contain an <embedded variable specification> or a <dynamic parameter specification>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.3 <assertion definition>

Function

Specify an integrity constraint by means of an assertion and specify when the assertion is to be checked.

Format

No additional Format items.

Syntax Rules

1) Augments SR4) The <search condition> shall also not contain an <embedded variable specification> or a <dynamic parameter specification>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.4 <trigger definition>

Function

Define triggered SQL-statements.

Format

No additional Format items.

Syntax Rules

1) Insert this SR The <triggered action> shall not contain a <dynamic parameter specification> or an <embedded variable name>.

2) Insert this SR The <triggered SQL statement> shall not generally contain an <SQL dynamic statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.5 <SQL-invoked routine>

Function

Define an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert before SR18)c) The <SQL routine body> shall not contain an <SQL dynamic statement>.

2) Insert this SR An <SQL-invoked routine> shall not contain a <dynamic parameter specification> or an <embedded variable name>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11 SQL-client modules

11.1 <SQL-client module definition>

Function

Define an SQL-client module.

Format

```
<module authorization clause> ::=
          SCHEMA <schema name>
        | AUTHORIZATION <module authorization identifier>
              [ FOR STATIC { ONLY | AND DYNAMIC } ]
        | SCHEMA <schema name> AUTHORIZATION <module authorization identifier>
              [ FOR STATIC { ONLY | AND DYNAMIC } ]

<module contents> ::=
          !! All alternatives from ISO/IEC 9075-2
        | <dynamic declare cursor>
```

Syntax Rules

1) Insert this SR A <dynamic declare cursor> shall precede in the text of the <SQL-client module definition> any <externally-invoked procedure> that references the <cursor name> of the <dynamic declare cursor>.

2) Insert this SR If neither FOR STATIC ONLY nor FOR STATIC AND DYNAMIC is specified, then FOR STATIC AND DYNAMIC is implicit.

Access Rules

No additional Access Rules.

General Rules

1) Augments GR5) After the last time that an SQL-agent performs a call of an <externally-invoked procedure>, following the effective execution of a <rollback statement> or a <commit statement>, a <deallocate descriptor statement> that specifies

   DEALLOCATE DESCRIPTOR *D*

   is effectively executed, where *D* is the <descriptor name> of any system descriptor area that is currently allocated within an SQL-session associated with the SQL-agent.

2) Insert this GR If FOR STATIC ONLY is specified, then a *shadow module* is effectively created to contain <externally-invoked procedure>s corresponding to each SQL-statement that is prepared or executed dynamically by <prepare statement>s or <execute immediate statement>s contained in this <SQL-client module definition>.

Conformance Rules

1) Insert this CR Without Feature B051, “Enhanced execution rights”, conforming SQL language shall not specify FOR STATIC ONLY or FOR STATIC AND DYNAMIC.

2) Insert this CR Without Feature B031, “Basic dynamic SQL”, a <module contents> shall not be a <dynamic declare cursor>.

11.2 Calls to an <externally-invoked procedure>

Function

Define a routine.

Format

No additional Format items.

Syntax Rules

1) Augments SR2)e) The base type of any host parameter shall be an Ada data type declared in an Ada package named Interfaces.SQL, as specified in ISO/IEC 9075-2, with the following additional constants:

```
ATTEMPT_TO_ASSIGN_TO_NON_UPDATABLE_COLUMN_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "0U000";
ATTEMPT_TO_ASSIGN_TO_ORDERING_COLUMN_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "0V000";
DYNAMIC_SQL_ERROR_NO_SUBCLASS:
        constant SQLSTATE_TYPE := "07000";
DYNAMIC_SQL_ERROR_CURSOR_SPECIFICATION_CANNOT_BE_EXECUTED:
        constant SQLSTATE_TYPE := "07003";
DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_COUNT:
        constant SQLSTATE_TYPE := "07008";
DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_INDEX:
        constant SQLSTATE_TYPE := "07009";
DYNAMIC_SQL_ERROR_PREPARED_STATEMENT_NOT_A_CURSOR_SPECIFICATION:
        constant SQLSTATE_TYPE := "07005";
DYNAMIC_SQL_ERROR_RESTRICTED_DATA_TYPE_ATTRIBUTE_VIOLATION:
        constant SQLSTATE_TYPE := "07006";
DYNAMIC_SQL_ERROR_DATA_TYPE_TRANSFORM_FUNCTION_VIOLATION:
        constant SQLSTATE_TYPE := "0700B";
DYNAMIC_SQL_ERROR_UNDEFINED_DATA_VALUE:
        constant SQLSTATE_TYPE := "0700C";
DYNAMIC_SQL_ERROR_UNDEFINED_DATA_TARGET:
        constant SQLSTATE_TYPE := "0700D";
DYNAMIC_SQL_ERROR_UNDEFINED_LEVEL_VALUE:
        constant SQLSTATE_TYPE := "0700E";
DYNAMIC_SQL_ERROR_UNDEFINED_DATETIME_INTERVAL_CODE:
        constant SQLSTATE_TYPE := "0700F";
DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_DYNAMIC_PARAMETER_SPEC:
        constant SQLSTATE_TYPE := "07001";
DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_TARGET_SPEC:
        constant SQLSTATE_TYPE := "07002";
DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_DYNAMIC_PARAMETERS:
        constant SQLSTATE_TYPE := "07004";
DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_RESULT_FIELDS:
        constant SQLSTATE_TYPE := "07007";
INVALID_CHARACTER_SET_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2C000";
INVALID_SCHEMA_NAME_LIST_SPECIFICATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0E000";
INVALID_SQL_INVOKED_PROCEDURE_REFERENCE_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0M000";
INVALID_TRANSFORM_GROUP_NAME_SPECIFICATION_NO_SUBCLASS:
```

```
                constant SQLSTATE_TYPE :="0S000";
        TARGET_TABLE_DISAGREES_WITH_CURSOR_SPECIFICATION_NO_SUBCLASS:
                constant SQLSTATE_TYPE :="0T000";
        WARNING_INSUFFICIENT_ITEM_DESCRIPTOR_AREAS:
                constant SQLSTATE_TYPE := "01005";
```

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.3 <SQL procedure statement>

Function

Define all of the SQL-statements that are <SQL procedure statement>s.

Format

```
<SQL executable statement> ::=
        !! All alternatives from ISO/IEC 9075-2
      | <SQL dynamic statement>

<SQL session statement> ::=
        !! All alternatives from ISO/IEC 9075-2
      | <set session characteristics statement>
      | <set catalog statement>
      | <set schema statement>
      | <set names statement>
      | <set path statement>
      | <set transform group statement>

<SQL dynamic statement> ::=
        <system descriptor statement>
      | <prepare statement>
      | <deallocate prepared statement>
      | <describe statement>
      | <execute statement>
      | <execute immediate statement>
      | <SQL dynamic data statement>

<SQL dynamic data statement> ::=
        <allocate cursor statement>
      | <dynamic open statement>
      | <dynamic fetch statement>
      | <dynamic close statement>
      | <dynamic delete statement: positioned>
      | <dynamic update statement: positioned>

<system descriptor statement> ::=
        <allocate descriptor statement>
      | <deallocate descriptor statement>
      | <set descriptor statement>
      | <get descriptor statement>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR2) If the non-dynamic or dynamic execution of an <SQL data statement>, <SQL dynamic data statement>, <dynamic select statement>, or <dynamic single row select statement> occurs within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement and this is not allowed by the SQL-implementation, then an exception condition is raised: *invalid transaction state — schema and data statement mixing not supported*.

2) Replace GR5)a)i)6) If *S* successfully initiated or resumed an SQL-session, then subsequent calls to an <externally-invoked procedure> and subsequent invocations of <direct SQL statement>s by the SQL-agent are associated with the SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

3) Replace GR5)a)iii)1)D) Subsequent calls to an <externally-invoked procedure> and subsequent invocations of <direct SQL statement>s by the SQL-agent are associated with the SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

4) Replace GR5)a)iii)3)A) An SQL-transaction is effectively initiated and associated with this call and with subsequent calls of any <externally-invoked procedure> by that SQL-agent and with this and subsequent invocations of <direct SQL statement>s by that SQL-agent until the SQL-agent terminates that SQL-transaction.

Conformance Rules

1) Insert this CR Without Feature B031, "Basic dynamic SQL", an <SQL procedure statement> shall not be an <SQL dynamic statement>.

12 Data manipulation

12.1 <select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR4) For each <target specification> *TS* that is an <embedded variable name>, then the Syntax Rules of Subclause 9.1, “Retrieval assignment”, shall apply to *TS* and the corresponding element of the <select list>, as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) Insert before GR6) If the <target specification> *TS* is an <embedded variable name>, then the first value in the row of *Q* is assigned to *TS* according to the General Rules of Subclause 9.1, “Retrieval assignment”, as *VALUE* and *TARGET*, respectively.

2) Insert after GR5) For each <target specification> *TS* that is an <embedded variable name>, the corresponding value in the row of *Q* is assigned to *TS* according to the General Rules of Subclause 9.1, “Retrieval assignment”, as *VALUE* and *TARGET*, respectively. The assignment of values to targets in the <select target list> is in an implementation-dependent order.

Conformance Rules

No additional Conformance Rules.

12.2 <free locator statement>

Function

Remove the association between a locator variable and the value that is represented by that locator.

Format

```
<locator reference> ::=
        !! All alternatives from ISO/IEC 9075-2
      | <host variable name>
```

Syntax Rules

1) Insert after SR1) Each host variable identified by the <host variable name> immediately contained in <locator reference> shall be a binary object locator variable, a character large object locator variable, an array locator variable, or a user-defined type locator variable.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

13 Transaction management

13.1 <commit statement>

Function

Terminate the current SQL-transaction with commit.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR The <statement name> or <extended statement name> of every held cursor remains valid.

Conformance Rules

No additional Conformance Rules.

14 Session management

14.1 <set catalog statement>

Function

Set the default catalog name for unqualified <schema name>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set catalog statement> ::=
      SET <catalog name characteristic>

<catalog name characteristic> ::=
      CATALOG <value specification>
```

Syntax Rules

1) The declared type of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) Let *S* be <value specification> and let *V* be the character string that is the value of

```
TRIM ( BOTH ' ' FROM S )
```

2) If *V* does not conform to the Format and Syntax Rules of a <catalog name>, then an exception condition is raised: *invalid catalog name*.

3) The default catalog name of the current SQL-session is set to *V*.

Conformance Rules

1) Without Feature F761, “Session management”, and Feature F651, “Catalog name qualifiers”, conforming SQL language shall not contain any <set catalog statement>.

14.2 <set schema statement>

Function

Set the default schema name for unqualified <schema qualified name>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set schema statement> ::=
      SET <schema name characteristic>

<schema name characteristic> ::=
      SCHEMA <value specification>
```

Syntax Rules

1) The declared type of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) Let *S* be <value specification> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

2) If *V* does not conform to the Format and Syntax Rules of a <schema name>, then an exception condition is raised: *invalid schema name.*

3) Case:

   a) If *V* conforms to the Format and Syntax Rules for a <schema name> that contains a <catalog name>, then let *X* be the <catalog name> part and let *Y* be the <unqualified schema name> part of *V*. The following statement is implicitly executed:

      SET CATALOG '*X*'

      and the <set schema statement> is effectively replaced by:

      SET SCHEMA '*Y*'

   b) Otherwise, the default unqualified schema name of the current SQL-session is set to *V*.

Conformance Rules

1) Without Feature F761, "Session management", conforming SQL language shall not contain any <set schema statement>.

14.3 <set names statement>

Function

Set the default character set name for <character string literal>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set names statement> ::=
      SET <character set name characteristic>

<character set name characteristic> ::=
      NAMES <value specification>
```

Syntax Rules

1) The declared type of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) Let *S* be <value specification> and let *V* be the character string that is the value of

```
TRIM ( BOTH ' ' FROM S )
```

2) If *V* does not conform to the Format and Syntax Rules of a <character set name>, then an exception condition is raised: *invalid character set name.*

3) The default character set name of the current SQL-session is set to *V*.

Conformance Rules

1) Without Feature F761, “Session management”, and either Feature F451, “Character set definition”, or Feature F461, “Named character sets”, conforming SQL language shall not contain any <set names statement>.

14.4 <set path statement>

Function

Set the SQL-path used to determine the subject routine of <routine invocation>s with unqualified <routine name>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s, respectively, that are invoked directly. The SQL-path remains the current SQL-path of the SQL-session until another SQL-path is successfully set.

Format

```
<set path statement> ::=
      SET <SQL-path characteristic>

<SQL-path characteristic> ::=
      PATH <value specification>
```

Syntax Rules

1) The declared type of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) Let *S* be <value specification> and let *V* be the character string that is the value of

```
TRIM ( BOTH ' ' FROM S )
```

   a) If *V* does not conform to the Format and Syntax Rules of a <schema name list>, then an exception condition is raised: *invalid schema name list specification*.

   b) The SQL-path of the current SQL-session is set to *V*.

NOTE 7 – A <set path statement> that is executed between a <prepare statement> and an <execute statement> has no effect on the prepared statement.

Conformance Rules

1) Without Feature S071, “SQL paths in function and type name resolution”, Conforming SQL language shall not contain any <set path statement>.

14.5 <set transform group statement>

Function

Set the group name that identifies the group of transform functions for mapping values of user-defined data types to predefined data types.

Format

```
<set transform group statement> ::=
      SET <transform group characteristic>

<transform group characteristic> ::=
        DEFAULT TRANSFORM GROUP <value specification>
      | TRANSFORM GROUP FOR TYPE <user-defined type> <value specification>
```

Syntax Rules

1) The data type of the <value specification> shall be an SQL character data type.

2) If "TRANSFORM GROUP FOR TYPE" is specified, then let *UDT* be the user-defined type identified by <user-defined type>.

Access Rules

None.

General Rules

1) Let *S* be <value specification> and let *V* be the character string that is the value of

   `TRIM ( BOTH ' ' FROM S )`

   a) If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid transform group name specification*.

   b) Case:

      i) If "TRANSFORM GROUP FOR TYPE" is specified, then the transform group name corresponding to all subtypes of *UDT* for the current SQL-session is set to *V*.

      ii) Otherwise, the default transform group name for the current SQL-session is set to *V*.

   NOTE 8 – A <set transform group statement> that is executed after a <prepare statement> has no effect on the prepared statement.

Conformance Rules

1) Without Feature S241, "Transform functions", conforming SQL language shall not contain any <set transform group statement>.

15 Dynamic SQL

15.1 Description of SQL descriptor areas

Function

Specify the identifiers, data types, and codes used in SQL item descriptor areas.

Syntax Rules

1) An SQL item descriptor area comprises the items specified in Table 3, "Data types of <key word>s used in SQL item descriptor areas".

2) An SQL descriptor area comprises the items specified in Table 2, "Data types of <key word>s used in the header of SQL descriptor areas", and one or more occurrences of an SQL item descriptor area.

3) Given an SQL item descriptor area *IDA* in which the value of LEVEL is N, the *immediately subordinate descriptor areas* of *IDA* are those SQL item descriptor areas in which the value of LEVEL is *N*+1 and whose position in the SQL descriptor area follows that of *IDA* and precedes that of any SQL item descriptor area in which the value of LEVEL is less than *N*+1.

   The *subordinate descriptor areas* of *IDA* are those SQL item descriptor areas that are immediately subordinate descriptor areas of *IDA* or that are subordinate descriptor areas of an SQL item descriptor area that is immediately subordinate to *IDA*.

4) Given a data type *DT* and its descriptor *DE*, the *immediately subordinate descriptors* of *DE* are defined to be:

   Case:

   a) If *DT* is a row type, then the field descriptors of the fields of *DT*. The *i*-th immediately subordinate descriptor is the descriptor of the *i*-th field of *DT*.

   b) If *DT* is a collection type, then the descriptor of the associated element type of *DT*.

   The *subordinate descriptors* of *DE* are those descriptors that are immediately subordinate descriptors of *DE* or that are subordinate descriptors of a descriptor that is immediately subordinate to *DE*.

5) Given a descriptor *DE*, let SDE_j represent its *j*-th immediately subordinate descriptor. There is an implied ordering of the subordinate descriptors of *DE*, such that:

   a) SDE_1 is in the first ordinal position.

   b) The ordinal position of SDE_{j+1} is *K*+*NS*+1, where *K* is the ordinal position of SDE_j and *NS* is the number of subordinate descriptors of SDE_j. The implicitly ordered subordinate descriptors of SDE_j occupy contiguous ordinal positions starting at position *K*+1.

6) An item descriptor area *IDA* is *valid* if and only if TYPE indicates a code defined in Table 4, "Codes used for SQL data types in Dynamic SQL", and one of the following is true:

Case:

a) TYPE indicates NUMERIC and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type.

b) TYPE indicates DECIMAL and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type.

c) TYPE indicates FLOAT and PRECISION is a valid precision value for the FLOAT data type.

d) TYPE indicates INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

e) TYPE indicates BOOLEAN.

f) TYPE indicates BIT or BIT VARYING and LENGTH is a valid length value for the BIT data type.

g) TYPE indicates CHARACTER, CHARACTER VARYING, or CHARACTER LARGE OBJECT, LENGTH is a valid length value for TYPE, and CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are a valid qualified character set name for TYPE.

h) TYPE indicates BINARY LARGE OBJECT and LENGTH is a valid length value for the BINARY LARGE OBJECT data type.

i) TYPE indicates CHARACTER LARGE OBJECT LOCATOR.

j) TYPE indicates BINARY LARGE OBJECT LOCATOR.

k) TYPE indicates USER-DEFINED TYPE LOCATOR and USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME are a valid qualified user-defined type name.

l) TYPE indicates a <datetime type>, DATETIME_INTERVAL_CODE is a code specified in Table 5, "Codes associated with datetime data types in Dynamic SQL", and PRECISION is a valid value for the <time precision> or <timestamp precision> of the indicated datetime data type.

m) TYPE indicates an <interval type>, DATETIME_INTERVAL_CODE is a code specified in Table 6, "Codes used for <interval qualifier>s in Dynamic SQL", and DATETIME_INTERVAL_PRECISION and PRECISION are valid values for <interval leading field precision> and <interval fractional seconds precision> for an <interval qualifier>.

n) TYPE indicates REF, LENGTH is the implementation-defined length in octets for the REF type, and USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME are a valid qualified user-defined type name, and SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME are a valid qualified table name.

o) TYPE indicates ROW, the value *N* of DEGREE is a valid value for the degree of a row type, there are exactly *N* immediately subordinate descriptor areas of *IDA* and those SQL item descriptor areas are valid.

p) TYPE indicates ARRAY or ARRAY LOCATOR, the value of CARDINALITY is a valid value for the cardinality of an array, there is exactly one immediately subordinate descriptor area of *IDA* and that SQL item descriptor area is valid.

q) TYPE indicates an implementation-defined data type.

7) The declared type *T* of a <simple value specification> or a <simple target specification> *SVT* is said to *match* the data type specified by a valid item descriptor area *IDA* if and only if one of the following conditions is true.

Case:

a) TYPE indicates NUMERIC and *T* is specified by NUMERIC(*P*,*S*), where *P* is the value of PRECISION and *S* is the value of SCALE.

b) TYPE indicates DECIMAL and *T* is specified by DECIMAL(*P*,*S*), where *P* is the value of PRECISION and *S* is the value of SCALE.

c) TYPE indicates INTEGER and *T* is specified by INTEGER.

d) TYPE indicates SMALLINT and *T* is specified by SMALLINT.

e) TYPE indicates FLOAT and *T* is specified by FLOAT(*P*), where *P* is the value of PRECISION.

f) TYPE indicates REAL and *T* is specified by REAL.

g) TYPE indicates DOUBLE PRECISION and *T* is specified by DOUBLE PRECISION.

h) TYPE indicates BOOLEAN and *T* is specified by BOOLEAN.

i) TYPE indicates BIT and *T* is specified by BIT(*L*), where *L* is the value of LENGTH.

j) TYPE indicates BIT VARYING and *T* is specified by BIT VARYING(*L*), where

Case:

i) *SVT* is a <simple value specification> and *L* is the value of LENGTH.

ii) *SVT* is a <simple target specification> and *L* is not less than the value of LENGTH.

k) TYPE indicates CHARACTER and *T* is specified by CHARACTER(*L*), where *L* is the value of LENGTH and the <character set specification> formed by the values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME identifies the character set of *SVT*.

l) Either TYPE indicates CHARACTER VARYING and *T* is specified by CHARACTER VARYING(*L*) or type indicates CHARACTER LARGE OBJECT and *T* is specified by CHARACTER LARGE OBJECT(*L*), where the <character set specification> formed by the values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME identifies the character set of *SVT* and

Case:

i) *SVT* is a <simple value specification> and *L* is the value of LENGTH.

ii) *SVT* is a <simple target specification> and *L* is not less than the value of LENGTH.

m) TYPE indicates BINARY LARGE OBJECT and *T* is specified by BINARY LARGE OBJECT(*L*) and

Case:

i) *STV* is a <simple value specification> and *L* is the value of LENGTH.

ii) *STV* is a <simple target specification> and *L* is not less than the value of LENGTH.

n) TYPE indicates CHARACTER LARGE OBJECT LOCATOR and *T* is specified by CHARACTER LARGE OBJECT LOCATOR.

o) TYPE indicates BINARY LARGE OBJECT LOCATOR and *T* is specified by BINARY LARGE OBJECT LOCATOR.

p) TYPE indicates USER-DEFINED TYPE and *T* is specified by USER-DEFINED TYPE LOCATOR, where the <user-defined type name> formed by the values of USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME identifies the associated user-defined type of *SVT*.

q) TYPE indicates REF and *T* is specified by REF, where the <user-defined type name> formed by the values of USER_DEFINED_DATA_TYPE_CATALOG, USER_DEFINED_DATA_TYPE_SCHEMA, and USER_DEFINED_DATA_TYPE_NAME identifies the row type of *SVT*, and SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME identify the scope of the reference type.

r) TYPE indicates ROW, and *T* is a row type with degree *D*, where *D* is the value of DEGREE, and the data type of the *i*-th field of *SVT* matches the data type specified by the *i*-th immediately subordinate descriptor area of *IDA*.

s) TYPE indicates ARRAY and *T* is an array type with cardinality *C* and the data type of the element type of *T* matches the immediately subordinate descriptor area of *IDA*, and

Case:

i) *SVT* is a <simple value specification> and *C* is the value of CARDINALITY.

ii) *SVT* is a <simple target specification> and *C* is not less than the value of CARDINALITY.

t) TYPE indicates ARRAY LOCATOR and *T* is an array locator type whose associated array type has cardinality *C* and the data type of the element type of the associated array type of *T* matches the immediately subordinate descriptor area of *IDA*, and

Case:

i) *SVT* is a <simple value specification> and *C* is the value of CARDINALITY.

ii) *SVT* is a <simple target specification> and *C* is not less than the value of CARDINALITY.

u) TYPE indicates a data type from Table 4, “Codes used for SQL data types in Dynamic SQL”, other than an implementation-defined data type and *T* satisfies the implementation-defined rules for matching that data type.

v) TYPE indicates an implementation-defined data type and *T* satisfies the implementation-defined rules for matching that data type.

8) A data type *DT* is said to be *represented* by an SQL item descriptor area if a <simply value specification> of type *DT* matches the SQL item descriptor area.

Table 2—Data types of <key word>s used in the header of SQL descriptor areas

<key word>	Data Type
COUNT	exact numeric with scale 0 (zero)
DYNAMIC_FUNCTION	character string with character set SQL_IDENTIFIER and length not less than 128 characters
DYNAMIC_FUNCTION_CODE	exact numeric with scale 0 (zero)
KEY_TYPE	exact numeric with scale 0 (zero)
TOP_LEVEL_COUNT	exact numeric with scale 0 (zero)

Table 3—Data types of <key word>s used in SQL item descriptor areas

<key word>	Data Type
CARDINALITY	exact numeric with scale 0 (zero)
CHARACTER_SET_CATALOG	character string with character set SQL_IDENTIFIER and length not less than 128 characters
CHARACTER_SET_NAME	character string with character set SQL_IDENTIFIER and length not less than 128 characters
CHARACTER_SET_SCHEMA	character string with character set SQL_IDENTIFIER and length not less than 128 characters
COLLATION_CATALOG	character string with character set SQL_IDENTIFIER and length not less than 128 characters
COLLATION_NAME	character string with character set SQL_IDENTIFIER and length not less than 128 characters
COLLATION_SCHEMA	character string with character set SQL_IDENTIFIER and length not less than 128 characters
DATA	matches the data type represented by the SQL item descriptor area
DATETIME_INTERVAL_CODE	exact numeric with scale 0 (zero)
DATETIME_INTERVAL_PRECISION	exact numeric with scale 0 (zero)
DEGREE	exact numeric with scale 0 (zero)
INDICATOR	exact numeric with scale 0 (zero)
KEY_MEMBER	exact numeric with scale 0 (zero)
LENGTH	exact numeric with scale 0 (zero)
LEVEL	exact numeric with scale 0 (zero)
NAME	character string with character set SQL_IDENTIFIER and length not less than 128 characters
NULLABLE	exact numeric with scale 0 (zero)
OCTET_LENGTH	exact numeric with scale 0 (zero)
PARAMETER_MODE	exact numeric with scale 0 (zero)

Table 3—Data types of <key word>s used in SQL item descriptor areas (Cont.)

<key word>	Data Type
PARAMETER_ORDINAL_ POSITION	exact numeric with scale 0 (zero)
PARAMETER_SPECIFIC_ CATALOG	character string with character set SQL_IDENTIFIER and length not less than 128 characters
PARAMETER_SPECIFIC_ NAME	character string with character set SQL_IDENTIFIER and length not less than 128 characters
PARAMETER_SPECIFIC_ SCHEMA	character string with character set SQL_IDENTIFIER and length not less than 128 characters
PRECISION	exact numeric with scale 0 (zero)
RETURNED_CARDINALITY	exact numeric with scale 0 (zero)
RETURNED_LENGTH	exact numeric with scale 0 (zero)
RETURNED_OCTET_LENGTH	exact numeric with scale 0 (zero)
SCALE	exact numeric with scale 0 (zero)
SCOPE_CATALOG	character string with character set SQL_IDENTIFIER and length not less than 128 characters
SCOPE_NAME	character string with character set SQL_IDENTIFIER and length not less than 128 characters
SCOPE_SCHEMA	character string with character set SQL_IDENTIFIER and length not less than 128 characters
TYPE	exact numeric with scale 0 (zero)
UNNAMED	exact numeric with scale 0 (zero)
USER_DEFINED_TYPE_ CATALOG	character string with character set SQL_IDENTIFIER and length not less than 128 characters
USER_DEFINED_TYPE_NAME	character string with character set SQL_IDENTIFIER and length not less than 128 characters
USER_DEFINED_TYPE_ SCHEMA	character string with character set SQL_IDENTIFIER and length not less than 128 characters

NOTE 9 – "Matches" and "represented by", as applied to the relationship between a data type and an SQL item descriptor area are defined in the Syntax Rules of this Subclause.

Access Rules

None.

General Rules

1) Table 4, "Codes used for SQL data types in Dynamic SQL", specifies the codes associated with the SQL data types.

Table 4—Codes used for SQL data types in Dynamic SQL

Data Type	Code
Implementation-defined data types	< 0 (zero)
ARRAY	50
ARRAY LOCATOR	51
BIT	14
BIT VARYING	15
BLOB	30
BLOB LOCATOR	31
BOOLEAN	16
CHARACTER	1 (one)
CHARACTER VARYING	12
CLOB	40
CLOB LOCATOR	41
DATE, TIME WITHOUT TIME ZONE, TIME WITH TIME ZONE, TIMESTAMP WITHOUT TIME ZONE, or TIMESTAMP WITH TIME ZONE	9
DECIMAL	3
DOUBLE PRECISION	8
FLOAT	6
INTEGER	4
INTERVAL	10
NUMERIC	2
REAL	7
SMALLINT	5
USER-DEFINED TYPE LOCATOR	18
ROW TYPE	19
REF	20
User-defined types	17

2) Table 5, “Codes associated with datetime data types in Dynamic SQL”, specifies the codes associated with the datetime data types.

Table 5—Codes associated with datetime data types in Dynamic SQL

Datetime Data Type	Code
DATE	1 (one)
TIME WITH TIME ZONE	4
TIME WITHOUT TIME ZONE	2
TIMESTAMP WITH TIME ZONE	5
TIMESTAMP WITHOUT TIME ZONE	3

3) Table 6, “Codes used for <interval qualifier>s in Dynamic SQL”, specifies the codes associated with <interval qualifier>s for interval data types.

Table 6—Codes used for <interval qualifier>s in Dynamic SQL

Datetime Qualifier	Code
DAY	3
DAY TO HOUR	8
DAY TO MINUTE	9
DAY TO SECOND	10
HOUR	4
HOUR TO MINUTE	11
HOUR TO SECOND	12
MINUTE	5
MINUTE TO SECOND	13
MONTH	2
SECOND	6
YEAR	1 (one)
YEAR TO MONTH	7

4) The value of DYNAMIC_FUNCTION is a character string that identifies the type of the prepared or executed SQL-statement. Table 9, “SQL-statement codes”, specifies the identifier of the SQL-statements.

5) The value of DYNAMIC_FUNCTION_CODE is a number that identifies the type of the prepared or executed SQL-statement. Table 9, “SQL-statement codes”, specifies the identifier of the SQL-statements.

6) Table 7, “Codes used for input/output SQL parameter modes in Dynamic SQL”, specifies the codes used for the PARAMETER_MODE item descriptor field when describing a <call statement>.

Table 7—Codes used for input/output SQL parameter modes in Dynamic SQL

Parameter mode	Code
PARAMETER_MODE_IN	1 (one)
PARAMETER_MODE_INOUT	2
PARAMETER_MODE_OUT	4

Conformance Rules

None.

15.2 <allocate descriptor statement>

Function

Allocate an SQL descriptor area.

Format

```
<allocate descriptor statement> ::=
      ALLOCATE [ SQL ] DESCRIPTOR <descriptor name> [ WITH MAX <occurrences> ]

<occurrences> ::= <simple value specification>
```

Syntax Rules

1) The declared type of <occurrences> shall be exact numeric with scale 0 (zero).

2) If WITH MAX <occurrences> is not specified, then an implementation-defined default value for <occurrences> that is greater than 0 (zero) is implicit.

Access Rules

None.

General Rules

1) Let *S* be the <simple value specification> that is immediately contained in <descriptor name> and let *V* be the character string that is the result of

```
TRIM ( BOTH ' ' FROM S )
```

If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid SQL descriptor name*.

2) The <allocate descriptor statement> allocates an SQL descriptor area whose name is *V* and whose scope is specified by the <scope option>. The descriptor area will have at least <occurrences> number of item descriptor areas. The value of LEVEL in each of the item descriptor areas is set to 0 (zero). The values of all other fields in the SQL descriptor area are initially undefined. If an SQL descriptor has already been allocated whose name is *V*, whose scope is specified by the <scope option>, and that has not yet been deallocated, then an exception condition is raised: *invalid SQL descriptor name*.

3) If <occurrences> is less than 1 (one) or is greater than an implementation-defined maximum value, then an exception condition is raised: *dynamic SQL error — invalid descriptor index*. The maximum number of SQL descriptor areas that can be allocated at one time is implementation-defined.

Conformance Rules

1) Without Feature B032, “Extended dynamic SQL”, an <occurrences> and a <descriptor name> shall be a <literal>.

2) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <allocate descriptor statement>.

15.3 <deallocate descriptor statement>

Function

Deallocate an SQL descriptor area.

Format

```
<deallocate descriptor statement> ::=
      DEALLOCATE [ SQL ] DESCRIPTOR <descriptor name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The <deallocate descriptor statement> deallocates an SQL descriptor area whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is specified by the <scope option>. If an SQL descriptor is not currently allocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is specified by the <scope option>, then an exception condition is raised: *invalid SQL descriptor name*.

Conformance Rules

1) Without Feature B032, "Extended dynamic SQL", a <descriptor name> shall be a <literal>.

2) Without Feature B031, "Basic dynamic SQL", conforming SQL language shall not contain any <deallocate descriptor statement>.

15.4 <get descriptor statement>

Function

Get information from an SQL descriptor area.

Format

```
<get descriptor statement> ::=
      GET [ SQL ] DESCRIPTOR <descriptor name> <get descriptor information>

<get descriptor information> ::=
        <get header information> [ { <comma> <get header information> }... ]
      | VALUE <item number>
          <get item information> [ { <comma> <get item information> }... ]

<get header information> ::=
      <simple target specification 1> <equals operator> <header item name>

<header item name> ::=
        COUNT
      | KEY_TYPE
      | DYNAMIC_FUNCTION
      | DYNAMIC_FUNCTION_CODE
      | TOP_LEVEL_COUNT

<get item information> ::=
      <simple target specification 2> <equals operator> <descriptor item name>

<item number> ::= <simple value specification>

<simple target specification 1> ::= <simple target specification>

<simple target specification 2> ::= <simple target specification>

<descriptor item name> ::=
        CARDINALITY
      | CHARACTER_SET_CATALOG
      | CHARACTER_SET_NAME
      | CHARACTER_SET_SCHEMA
      | COLLATION_CATALOG
      | COLLATION_NAME
      | COLLATION_SCHEMA
      | DATA
      | DATETIME_INTERVAL_CODE
      | DATETIME_INTERVAL_PRECISION
      | DEGREE
      | INDICATOR
      | KEY_MEMBER
      | LENGTH
      | LEVEL
      | NAME
      | NULLABLE
      | OCTET_LENGTH
      | PARAMETER_MODE
      | PARAMETER_ORDINAL_POSITION
      | PARAMETER_SPECIFIC_CATALOG
      | PARAMETER_SPECIFIC_NAME
      | PARAMETER_SPECIFIC_SCHEMA
      | PRECISION
```

```
    | RETURNED_CARDINALITY
    | RETURNED_LENGTH
    | RETURNED_OCTET_LENGTH
    | SCALE
    | SCOPE_CATALOG
    | SCOPE_NAME
    | SCOPE_SCHEMA
    | TYPE
    | UNNAMED
    | USER_DEFINED_TYPE_CATALOG
    | USER_DEFINED_TYPE_NAME
    | USER_DEFINED_TYPE_SCHEMA
```

Syntax Rules

1) The declared type of <item number> shall be exact numeric with scale 0 (zero).

2) For each <get header information>, the declared type of <simple target specification 1> shall be that shown in the Data Type column of the row in Table 2, “Data types of <key word>s used in the header of SQL descriptor areas”, whose <key word> column value is equivalent to <header item name>.

3) For each <get item information>, the declared type of <simple target specification 2> shall be that shown in the Data Type column of the row in Table 3, “Data types of <key word>s used in SQL item descriptor areas”, whose <key word> column value is equivalent to <descriptor item name>.

Access Rules

None.

General Rules

1) If a <descriptor name> specified in a <get descriptor statement> identifies an SQL descriptor area that is not currently allocated, then an exception condition is raised: *invalid SQL descriptor name*.

2) If the <item number> specified in a <get descriptor statement> is greater than the value of <occurrences> specified when the <descriptor name> was allocated or less than 1 (one), then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

3) If the <item number> specified in a <get descriptor statement> is greater than the value of COUNT, then a completion condition is raised: *no data*.

4) If the declared type of the <simple target specification> associated with the keyword DATA does not match the data type represented by the item descriptor area, then an exception condition is raised: *data exception — error in assignment*.

   NOTE 10 – “Match” and “represented by” are defined in the Syntax Rules of Subclause 15.1, “Description of SQL descriptor areas”.

5) Let *i* be the value of the <item number> contained in <get descriptor information>. Let *IDA* be the *i*-th <item descriptor area>. If a <get item information> specifies DATA, then:

   a) If *IDA* is subordinate to an item descriptor area whose TYPE field indicates ARRAY or ARRAY LOCATOR, then an exception condition is raised: *dynamic SQL error — undefined DATA value.*

   b) If the value of TYPE in *IDA* indicates ROW, then an exception condition is raised: *dynamic SQL error — undefined DATA value.*

   c) If the value of INDICATOR is negative and no <get item information> specifies INDICATOR, then an exception condition is raised: *data exception — null value, no indicator parameter*.

6) If an exception condition is raised in a <get descriptor statement>, then the values of all targets specified by <simple target specification 1> and <simple target specification 2> are implementation-dependent.

7) A <get descriptor statement> retrieves values from the SQL descriptor area and item specified by <descriptor name>. For each item, the value that is retrieved is the one established by the most recently executed <allocate descriptor statement>, <set descriptor statement>, or <describe statement> that references the specified SQL descriptor area and item. The value retrieved by a <get descriptor statement> for any field whose value is undefined is implementation-dependent.

   Case:

   a) If <get descriptor information> contains one or more <get header information>s, then for each <get header information> specified, the value of <simple target specification 1> is set to the value *V* in the SQL descriptor area of the field identified by the <header item name> by applying the General Rules of Subclause 9.2, "Store assignment", to <simple target specification 1> and *V* as *TARGET* and *VALUE*, respectively.

   b) If <get descriptor information> contains one or more <get item information>s, then:

      i) Let *i* be the value of the <item number> contained in the <get descriptor information>.

      ii) For each <get item information> specified, the value of <simple target specification 2> is set to the value *V* in the *i*-th SQL item descriptor area of the field identified by the <descriptor item name> by applying the General Rules of Subclause 9.2, "Store assignment", to <simple target specification 2> and *V* as *TARGET* and *VALUE*, respectively.

Conformance Rules

1) Without Feature B032, “Extended dynamic SQL”, a <descriptor name> shall be a <literal>.

2) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <get descriptor statement>.

3) Without Feature T301, “Functional dependencies”, and without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not specify KEY_MEMBER.

15.5 <set descriptor statement>

Function

Set information in an SQL descriptor area.

Format

```
<set descriptor statement> ::=
      SET [ SQL ] DESCRIPTOR <descriptor name> <set descriptor information>

<set descriptor information> ::=
        <set header information> [ { <comma> <set header information> }... ]
      | VALUE <item number>
          <set item information> [ { <comma> <set item information> }... ]

<set header information> ::=
      <header item name> <equals operator> <simple value specification 1>

<set item information> ::=
      <descriptor item name> <equals operator> <simple value specification 2>

<simple value specification 1> ::= <simple value specification>

<simple value specification 2> ::= <simple value specification>

<item number> ::= <simple value specification>
```

Syntax Rules

1) The declared type of <item number> shall be exact numeric with scale 0 (zero).

2) For each <set header information>, <header item name> shall not be KEY_TYPE, TOP_LEVEL_COUNT, DYNAMIC_FUNCTION, or DYNAMIC_FUNCTION_CODE, and the declared type of <simple value specification 1> shall be that in the Data Type column of the row of Table 2, "Data types of <key word>s used in the header of SQL descriptor areas", whose <key word> column value is equivalent to <header item name>.

3) For each <set information item>, the value of <descriptor item name> shall not be RETURNED_LENGTH, RETURNED_OCTET_LENGTH, RETURNED_CARDINALITY, OCTET_LENGTH, NULLABLE, KEY_MEMBER, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, NAME, UNNAMED, PARAMETER_MODE, PARAMETER_ORDINAL_POSITION, PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_SCHEMA, or PARAMETER_SPECIFIC_NAME. Other alternatives for <descriptor item name> shall not be specified more than once in a <set descriptor statement>. The declared type of <simple value specification 2> shall be that shown in the Data Type column of the row in Table 3, "Data types of <key word>s used in SQL item descriptor areas", whose <key word> column value is equivalent to <descriptor item name>.

4) If the <descriptor item name> specifies DATA, then <simple value specification 2> shall not be a <literal>.

Access Rules

None.

General Rules

1) If a <descriptor name> specified in a <set descriptor statement> identifies an SQL descriptor area that is not currently allocated, then an exception condition is raised: *invalid SQL descriptor name*.

2) If the <item number> specified in a <set descriptor statement> is greater than the value of <occurrences> specified when the <descriptor name> was allocated or less than 1 (one), then an exception condition is raised: *dynamic SQL error — invalid descriptor index*.

3) When more than one value is set in a single <set descriptor statement>, the values are effectively assigned in the following order: LEVEL, TYPE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME, LENGTH, INDICATOR, DEGREE, CARDINALITY, and DATA.

   When any value other than DATA is set, the value of DATA becomes undefined.

4) For every <set item information> specified, let *DIN* be the <descriptor item name>, let *V* be the value of the <simple value specification 2>, let *N* be the value of <item number>, and let *IDA* be the *N*-th item descriptor area.

   a) If *DIN* is DATA, then:

      i) If *IDA* is subordinate to an item descriptor area whose TYPE field indicates ARRAY or ARRAY LOCATOR, then an exception condition is raised: *dynamic SQL error — invalid DATA target*.

      ii) If TYPE in *IDA* indicates ROW, then an exception condition is raised: *dynamic SQL error — invalid DATA target*.

      iii) If the most specific type of *V* does not match the data type specified by the item descriptor area, then an exception condition is raised: *data exception — error in assignment*.

      NOTE 11 – "Match" is defined in the Syntax Rules of Subclause 15.1, "Description of SQL descriptor areas".

      iv) The value of DATA in *IDA* is set to *V*.

   b) If *DIN* is LEVEL, then:

      i) If *N* is 1 (one) and *V* is not 0 (zero), then an exception condition is raised: *dynamic SQL error — invalid LEVEL value*.

ii) If N is greater than 1 (one), then let *PIDA* be *IDA*'s immediately preceding item descriptor area and let K be its LEVEL value.

1) If $V = K$+1 and TYPE in *PIDA* does not indicate ROW, ARRAY, or ARRAY LOCATOR, then an exception condition is raised: *dynamic SQL error — invalid LEVEL value.*

2) If $V > K$+1, then an exception condition is raised: *dynamic SQL error — invalid LEVEL value.*

3) If $V < K$+1, then let $OIDA_i$ be the i-th item descriptor area to which *PIDA* is subordinate and whose TYPE field indicates ROW, let NS_i be the number of immediately subordinate descriptor areas of $OIDA_i$ between $OIDA_i$ and *IDA* and let D_i be the value of DEGREE in $OIDA_i$.

A) For each $OIDA_i$ whose LEVEL value is greater than V, if D_i is not equal to NS_i, then an exception condition is raised: *dynamic SQL error — invalid LEVEL value.*

B) If K is not 0 (zero), then let $OIDA_j$ be the $OIDA_i$ whose LEVEL value is K. If there exists no such $OIDA_j$ or D_j is not greater than NS_j, then an exception condition is raised: *dynamic SQL error — invalid LEVEL value.*

iii) The value of LEVEL in *IDA* is set to V.

c) If *DIN* is TYPE, then:

i) The value of TYPE in *IDA* is set to V.

ii) The value of all fields other than TYPE and LEVEL in *IDA* are set to implementation-dependent values.

iii) Case:

1) If V indicates CHARACTER, CHARACTER VARYING, or CHARACTER LARGE OBJECT, then CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA and CHARACTER_SET_NAME in *IDA* are set to the values for the default character set name for the SQL-session and LENGTH in *IDA* is set to 1 (one).

2) If V indicates CHARACTER LARGE OBJECT LOCATOR, then LENGTH in *IDA* is set to 1 (one).

3) If V indicates BIT or BIT VARYING, then LENGTH in *IDA* is set to 1 (one).

4) If V indicates BINARY LARGE OBJECT, then LENGTH in *IDA* is set to 1 (one).

5) If V indicates BINARY LARGE OBJECT LOCATOR, then LENGTH in *IDA* is set to 1 (one).

6) If V indicates DATETIME, then PRECISION in *IDA* is set to 0 (zero).

7) If V indicates INTERVAL, then DATETIME_INTERVAL_PRECISION in *IDA* is set to 2.

8) If V indicates NUMERIC or DECIMAL, then SCALE in *IDA* is set to 0 (zero) and PRECISION in *IDA* is set to the implementation-defined default value for the precision of NUMERIC or DECIMAL data types, respectively.

9) If *V* indicates FLOAT, then PRECISION in *IDA* is set to the implementation-defined default value for the precision of the FLOAT data type.

d) If *DIN* is DATETIME_INTERVAL_CODE, then

Case:

i) If TYPE in *IDA* indicates DATETIME, then

Case:

1) If *V* indicates DATE, TIME, or TIME WITH TIME ZONE, then PRECISION in *IDA* is set to 0 (zero) and DATETIME_INTERVAL_CODE in *IDA* is set to *V*.

2) If *V* indicates TIMESTAMP or TIMESTAMP WITH TIME ZONE, then PRECISION in *IDA* is set to 6 and DATETIME_INTERVAL_CODE in *IDA* is set to *V*.

3) Otherwise, an exception condition is raised: *dynamic SQL error — invalid DATETIME_INTERVAL_CODE*.

ii) If TYPE in *IDA* indicates INTERVAL, then

Case:

1) If *V* indicates DAY TO SECOND, HOUR TO SECOND, MINUTE TO SECOND, or SECOND, then PRECISION in *IDA* is set to 6, DATETIME_INTERVAL_PRECISION in *IDA* is set to 2 and DATETIME_INTERVAL_CODE in *IDA* is set to *V*.

2) If *V* indicates YEAR, MONTH, DAY, HOUR, MINUTE, YEAR TO MONTH, DAY TO HOUR, DAY TO MINUTE, or HOUR TO MINUTE, then PRECISION in *IDA* is set to 0 (zero), DATETIME_INTERVAL_PRECISION in *IDA* is set to 2 and DATETIME_INTERVAL_CODE in *IDA* is set to *V*.

3) Otherwise, an exception condition is raised: *dynamic SQL error — invalid DATETIME_INTERVAL_CODE*.

iii) Otherwise, an exception condition is raised: *dynamic SQL error — invalid DATETIME_INTERVAL_CODE*.

e) Otherwise, the value of *DIN* in IDA is set to *V* by applying the General Rules of Subclause 9.2, "Store assignment", to the field of *IDA* identified by *DIN* and *V* as *TARGET* and *VALUE*, respectively. .

5) For each <set header information> specified, the value of the field identified by <header item name> is set to the value *V* of <simple value specification 1> by applying the General Rules of Subclause 9.2, "Store assignment", to the field identified by the <header item name> and *V* as *TARGET* and *VALUE*, respectively.

6) If an exception condition is raised in a <set descriptor statement>, then the values of all elements of the item descriptor area specified in the <set descriptor statement> are implementation-dependent.

7) Restrictions on changing TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_

SCHEMA, and SCOPE_NAME values resulting from the execution of a <describe statement> before execution of an <execute statement>, <dynamic open statement>, or <dynamic fetch statement> are implementation-defined.

Conformance Rules

1) Without Feature B032, “Extended dynamic SQL”, a <descriptor name> shall be a <literal>.

2) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <set descriptor statement>.

15.6 <prepare statement>

Function

Prepare a statement for execution.

Format

```
<prepare statement> ::=
      PREPARE <SQL statement name> FROM <SQL statement variable>

<SQL statement variable> ::= <simple value specification>

<preparable statement> ::=
        <preparable SQL data statement>
      | <preparable SQL schema statement>
      | <preparable SQL transaction statement>
      | <preparable SQL control statement>
      | <preparable SQL session statement>
      | <preparable implementation-defined statement>

<preparable SQL data statement> ::=
        <delete statement: searched>
      | <dynamic single row select statement>
      | <insert statement>
      | <dynamic select statement>
      | <update statement: searched>
      | <preparable dynamic delete statement: positioned>
      | <preparable dynamic update statement: positioned>

<preparable SQL schema statement> ::=
        <SQL schema statement>

<preparable SQL transaction statement> ::=
        <SQL transaction statement>

<preparable SQL control statement> ::=
      <SQL control statement>

<preparable SQL session statement> ::=
        <SQL session statement>

<dynamic select statement> ::= <cursor specification>

<preparable implementation-defined statement> ::= !! See the Syntax Rules.
```

Syntax Rules

1) The <simple value specification> of <SQL statement variable> shall not be a <literal>.

2) The declared type of <SQL statement variable> shall be character string.

3) The Format and Syntax Rules for <preparable implementation-defined statement> are implementation-defined.

4) A <preparable SQL control statement> shall not contain an <SQL procedure statement> that is not a <preparable statement>, nor shall it contain a <dynamic single row select statement> or a <dynamic select statement>.

Access Rules

None.

General Rules

1) Let *P* be the contents of the <SQL statement variable>. If *P* is an <SQL control statement>, then let *PS* be an <SQL procedure statement> contained in *P*.

2) Two subfields *SF1* and *SF2* of row types *RT1* and *RT2* are *corresponding subfields* if either *SF1* or *SF2* are positionally corresponding fields of *RT1* and *RT2*, respectively, or *SF1* and *SF2* are positionally corresponding fields of *RT1SF1* and *RT2SF2* and *RT1SF1* and *RT2SF2* are the declared types of corresponding subfields of *RT1* and *RT2* respectively.

3) If *P* does not conform to the Format, Syntax Rules, and Access Rules of a <preparable statement>, or if *P* contains a <simple comment> then an exception condition is raised: *syntax error or access rule violation*.

4) Let *DTGN* be the default transform group name and let *TFL* be the list of {user-defined type name — transform group name} pairs used to identify the group of transform functions for every user-defined type that is referenced in *P*. *DTGN* and *TFL* are not affected by the execution of a <set transform group statement> after *P* is prepared.

5) Let *DPV* be a <value expression> that is either a <dynamic parameter specification> or a <dynamic parameter specification> immediately contained in any number of <left paren> <right paren> pairs. Initially, the declared type of such a <value expression> is, by definition, undefined. A data type is *undefined* if it is neither a data type defined in this standard nor a data type defined by the implementation.

6) Let *MP* be the implementation-defined maximum value of <precision> for the NUMERIC data type. Let *ML* be the implementation-defined maximum value of <length> for the CHARACTER VARYING data type. For each <value expression> *DP* in *P* or *PS* that meets the criteria for *DPV* let *DT* denote its declared type.

   a) Case:

      i) If *DP* is immediately followed by an <interval qualifier> *IQ*, then *DT* is INTERVAL *IQ*.

      ii) If *DP* is the <numeric value expression> simply contained in an <element reference>, then *DT* is NUMERIC (*MP*,0).

      iii) If *DP* is the <string value expression> simply contained in a <char length expression> or an <octet length expression>, then *DT* is CHARACTER VARYING(*ML*) with an implementation-defined character set.

      iv) If *DP* is the <string value expression> simply contained in a <bit length expression>, then *DT* is BIT VARYING(*L*) where *L* is the implementation-defined maximum value of length for BIT VARYING.

v) If *DP* is either the <numeric value expression dividend> *X1* or the <numeric value expression divisor> *X2* simply contained in a <modulus expression>, then if *DP* is *X1* (*X2*), then *DT* is the declared type of *X2* (*X1*).

vi) If *DP* is either *X1* or *X2* in a <position expression> of the form "POSITION <left paren> *X1* IN *X2* <right paren>", and if *DP* is *X1* (*X2*), then

Case:

1) If the declared type of *X2* (*X1*) is CHARACTER or CHARACTER VARYING with character set *CS*, then *DT* is CHARACTER VARYING (*ML*) with character set *CS*.

2) Otherwise, *DT* is the declared type of *X2* (*X1*).

vii) If *DP* is either *X2* or *X3* in a <string value function> of the form "SUBSTRING <left paren> *X1* FROM *X2* FOR *X3* <right paren>" or "SUBSTRING <left paren> *X1* FROM *X2* <right paren>", then *DT* is NUMERIC (*MP*, 0).

viii) If *DP* is any of *X1*, *X2*, *X3*, or X4 in a <string value function> of the form "OVERLAY <left paren> *X1* PLACING *X2* FROM *X3* FOR *X4* <right paren>" or "OVERLAY <left paren> *X1* PLACING *X2* FROM *X3* <right paren>", then

Case:

1) If *DP* is *X1* (*X2*), then

Case:

A) If the declared type of *X2* (*X1*) is CHARACTER or CHARACTER VARYING with character set *CS*, *DT* is CHARACTER VARYING (*ML*) with character set *CS*.

B) Otherwise, *DT* is the declared type of *X2* (*X1*).

2) Otherwise, *DT* is NUMERIC (*MP*, 0).

ix) If *DP* is either *X1* or *X2* in a <value expression> of the form "*X1* <concatenation operator> *X2*" and *DP* is *X1* (*X2*), then

Case:

1) If the declared type of *X2* (*X1*) is CHARACTER or CHARACTER VARYING with character set *CS*, then *DT* is CHARACTER VARYING (*ML*) with character set *CS*.

2) Otherwise, *DT* is the declared type of *X2* (*X1*).

x) If *DP* is either *X1* or *X2* in a <value expression> of the form "*X1* <asterisk> *X2*" or "*X1* <solidus> *X2*" and *DP* is *X1* (*X2*), then

Case:

1) If *DP* is *X1*, then *DT* is the declared type of *X2*.

2) Otherwise,

Case:

A) If the declared type of *X1* is an interval type, then *DT* is NUMERIC (*MP*, 0).

B) Otherwise, *DT* is the declared type of *X2* (*X1*).

xi) If *DP* is either *X1* or *X2* in a <value expression> of the form "*X1* <plus sign> *X2*" or "*X1* <minus sign> *X2*", then

Case:

1) If *DP* is *X1* in an expression of the form "*X1* <minus sign> *X2*", then *DT* is the declared type of *X2*.

2) Otherwise, if *DP* is *X1* (*X2*), then

Case:

A) If the declared type of *X2* (*X1*) is date, then *DT* is INTERVAL YEAR (*PR*) TO MONTH, where *PR* is the implementation-defined maximum <interval leading field precision>.

B) If the declared type of *X2* (*X1*) is time or timestamp, then *DT* is INTERVAL DAY (*PR*) TO SECOND(*FR*), where *PR* and *FR* are the implementation-defined maximum <interval leading field precision> and maximum <interval fractional seconds precision>, respectively.

C) Otherwise, *DT* is the declared type of *X2* (*X1*).

xii) If *DP* is the <value expression primary> simply contained in a <boolean primary>, then *DT* is BOOLEAN.

xiii) If *DP* is an <array element> simply contained in an <array element list> *AEL* or *DP* represents the value of a subfield *SF* of the declared type of an <array element> simply contained in an <array element list> *AEL*, then let *ET* be the result of applying the Syntax Rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of the <array element>s simply contained in *AEL*.

Case:

1) If *DP* is an <array element> of *AEL*, then *DT* is *ET*.

2) Otherwise, *DT* is the declared type of the subfield of *ET* that corresponds to *SF*.

xiv) If *DP* is the <cast operand> simply contained in a <cast specification> *CS* or *DP* represents the value of a subfield *SF* of the declared type of the <cast operand> simply contained in a <cast specification> *CS*, then let *CT* be the simply contained <cast target> of *CS*.

Case:

1) Let *RT* be a data type determined as follows:

Case:

A) If *CT* immediately contains ARRAY, then *RT* is undefined.

B) If *CT* immediately contains <data type>, then *RT* is that data type.

C) If *CT* simply contains <domain name> *D*, then *RT* is the declared type of the domain identified by *D*.

2) Case:

A) If *DP* is the <cast operand> of *CS*, *DT* is *RT*.

B) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xv) If *DP* is a <value expression> simply contained in a <case abbreviation> *CA* or *DP* represents the value of a subfield, SF, of the declared type of such a <value expression>, then let *RT* be the result of applying the Syntax Rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of the <value expression>s simply contained in *CA*.

Case:

1) If *DP* is a <value expression> simply contained in *CA*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xvi) If *DP* is a <result expression> simply contained in a <case specification> *CE* or *DP* represents the value of a subfield *SF* of the declared type of such a <result expression>, then let *RT* be the result of applying the Syntax Rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of the <result expression>s simply contained in *CE*.

Case:

1) If *DP* is a <result expression> simply contained in *CE*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xvii) If *DP* is a <case operand> or <when operand> simply contained in a <simple case> *CE* or *DP* represents the value of a subfield, SF, of the declared type of such a <case operand> or <when operand>, then *RT* is the result of applying the Syntax Rules of Subclause 9.3, to the declared types of the <case operand> and <when operand>s simply contained in *CE*.

Case:

1) If *DP* is a <case operand> or <when operand> simply contained in *CE*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xviii) If *DP* is a <row value expression> or <contextually typed row value expression> simply contained in a <table value constructor> or <contextually typed table value constructor> *TVC*, or if *DP* represents the value of a subfield *SF* of the declared type of such a <row value expression> or <contextually typed row value expression>, then

Case:

1) Let *RT* be a data type determined as follows.

Case:

A) If *TVC* is simply contained in a <query expression> that is simply contained in an <insert statement> *IS* or if *TVC* is immediately contained in the <insert column and source> of an <insert statement> *IS*, then *RT* is a row type in which the declared type of the *i*-th field is the declared type of the *i*-th column in the explicit or implicit <insert column list> of *IS* and the degree of *RT* is equal to the number of columns in the explicit or implicit <insert column list> of *IS*.

B) Otherwise, *RT* is the result of applying the Syntax Rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of the <row value expression>s or <contextually typed row value expression>s simply contained in *TVC*.

2) Case:

A) If *DP* is a <row value expression> or <contextually typed row value expression> simply contained in *TVC*, then *DT* is *RT*.

B) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xix) If *DP* is a <row value expression> simply contained in a <comparison predicate>, <distinct predicate> or <between predicate> *PR* or if *DP* represents the value of a subfield, SF, of the declared type of such a <row value expression>, then let *RT* be the result of applying the Syntax Rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of the <row value expression>s simply contained in *PR*.

Case:

1) If *DP* is a <row value expression> simply contained in *PR*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xx) If *DP* is a <row value expression> simply contained in a <quantified comparison predicate> or <match predicate> *PR* or *DP* represents the value of a subfield *SF* of the declared type of such a <row value expression>, then let *RT* be the declared type of the <table subquery> simply contained in *PR*.

Case:

1) If *DP* is a <row value expression> simply contained in *PR*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xxi) If *DP* is a <row value expression> simply contained in an <in predicate> *PR* or if *DP* represents the value of a subfield *SF* of the declared type of such a <row value expression>, then let *RT* be the result of applying the Syntax Rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of the <row value expression>s simply contained in *PR* and the declared row type of the <table subquery> (if any) simply contained in *PR*.

Case:

1) If *DP* is a <row value expression> simply contained in *PR*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xxii) If *DP* is the first <row value constructor element> simply contained in either <row value expression 1> *RV1* or <row value expression 2> *RV2* in an <overlaps predicate> *PR*, then

Case:

1) If both *RV1* and *RV2* simply contain a <row value constructor> whose first <row value constructor element> meets the criteria for *DPV*, then *DT* is TIMESTAMP WITH TIME ZONE.

2) Otherwise, if *DP* is simply contained in *RV1* (*RV2*), then *DT* is the declared type of the first <row value constructor element> of *RV2* (*RV1*).

xxiii) If *DP* is simply contained in a <like predicate> or a <similar predicate> *PR*, then let *X1* represent the <character match value> or the <octet match value>, let *X2* represent the <character pattern>, the <octet pattern> or the <similar pattern>, and let *X3* represent the <escape character> or the <escape octet>.

Case:

1) If all *X1*, *X2* and *X3* meet the criteria for *DPV*, then *DT* is CHARACTER VARYING (*ML*) with an implementation-defined character set.

2) Otherwise, let *RT* be the result of applying the Syntax Rules of Subclause 9.3, "Data types of results of aggregations", to the declared types of *X1*, *X2* and *X3*.

Case:

A) If *RT* is CHARACTER or CHARACTER VARYING with character set *CS*, then *DT* is CHARACTER VARYING(*ML*) with character set *CS*.

B) Otherwise, *DT* is *RT*.

xxiv) If *DP* is the <value expression> simply contained in an <update source> of a <set clause> *SC* or if *DP* represents the value of a subfield *SF* of the declared type of such a <value expression>, then let *RT* be the declared type of the <update target> or <mutated set clause> specified in *SC*.

Case:

1) If *DP* is the <value expression> simply contained in *SC*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

xxv) If *DP* is the <value specification> immediately contained in a <catalog name characteristic>, <schema name characteristic>, <character set name characteristic>, <SQL-path characteristic>, <transform group characteristic>, <role specification> or <set session authorization identifier statement>, then *DT* is CHARACTER VARYING (*ML*) with an implementation-defined character set.

xxvi) If *DP* is the <interval value expression> immediately contained in a <set local time zone statement>, then *DT* is INTERVAL HOUR TO MINUTE.

xxvii) If *DP* is an <SQL argument> of a <routine invocation> *RI* immediately contained in a <call statement> or if *DP* is the value of a subfield *SF* of the declared type of a <value expression> immediately contained in such an <SQL argument>, and if *DP* is the *i*-th <SQL argument> of *RI* or is contained in the *i*-th <SQL argument> of *RI*, then let *RT* denote the declared type of the *i*-th SQL parameter of the subject routine of *RI* determined by applying the Syntax Rules of Subclause 8.1, "<routine invocation>", to *RI*.

Case:

1) If *DP* is the *i*-th <SQL argument> of *RI*, then *DT* is *RT*.

2) Otherwise, *DT* is the declared type of the subfield of *RT* that corresponds to *SF*.

b) If *DT* is undefined, then an exception condition is raised: *syntax error or access rule violation*.

7) Whether a <dynamic parameter specification> is an input argument, an output argument, or both an input and an output argument is determined as follows.

   Case:

   a) If P is a <call statement>, then:

      i) Let SR be the subject routine of the <routine invocation> RI immediately contained in P. Let n be the number of <SQL argument>s in the <SQL argument list> immediately contained in RI.

      ii) Let A_y, 1 (one) $\leq y \leq n$, be the y-th <SQL argument> of the <SQL argument list> immediately contained in RI.

      iii) For each <dynamic parameter specification> D contained in some <SQL argument> A_k, 1 (one) $\leq k \leq n$:

         1) D is an input <dynamic parameter specification> if the <SQL parameter mode> of the k-th SQL parameter of SR is INPUT or INOUT.

         2) D is an output <dynamic parameter specification> if the <SQL parameter mode> of the k-th SQL parameter of SR is OUTPUT or INOUT.

   b) Otherwise:

      i) If a <dynamic parameter specification> is contained in a <target specification>, then it is an output <dynamic parameter specification>.

      ii) If a <dynamic parameter specification> is contained in a <value specification>, then it is an input <dynamic parameter specification>.

8) If P or PS is a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, then let CN be the <cursor name> contained in P or PS, respectively.

   Case:

   a) If P or PS contains a <scope option> that specifies GLOBAL, then

      Case:

      i) If there exists an extended dynamic cursor EDC with an <extended cursor name> having a global scope and a <cursor name> that is equivalent to CN, then EDC is the cursor referenced by P or PS.

      ii) Otherwise, an exception condition is raised: *invalid cursor name.*

   b) If P or PS contains a <scope option> that specifies LOCAL, or if no <scope option> is specified, then the *potentially referenced cursors* of P or PS include every declared dynamic cursor whose <cursor name> is equivalent to CN and whose scope is the containing module and every extended dynamic cursor having an <extended cursor name> that has a scope of the containing module and whose <cursor name> is equivalent to CN.

      Case:

      i) If the number of potentially referenced cursors is greater than 1 (one), then an exception condition is raised: *ambiguous cursor name.*

ii) If the number of potentially referenced cursors is less than 1 (one), then an exception condition is raised: *invalid cursor name.*

iii) Otherwise, *CN* refers to the single potentially referenced cursor of *P*.

9) If <extended statement name> is specified for the <SQL statement name>, then let *S* be <simple target specification> and let *V* be the character string that is the result of

```
TRIM ( BOTH ' ' FROM S )
```

If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid SQL statement identifier*.

10) If <statement name> is specified for the <SQL statement name>, *P* is not a <cursor specification>, and <statement name> is associated with a cursor *C* through a <dynamic declare cursor>, then an exception condition is raised: *dynamic SQL error — prepared statement not a cursor specification*.

11) If the value of the <SQL statement name> identifies an existing prepared statement, then an implicit

```
DEALLOCATE PREPARE SSN
```

is executed, where *SSN* is the value of the <SQL statement name>.

12) *P* is prepared for execution.

13) Case:

a) If <extended statement name> is specified for the <SQL statement name>, then the value of the <extended statement name> is associated with the prepared statement. This value and explicit or implied <scope option> shall be specified for each <execute statement> or <allocate cursor statement> that is to be associated with this prepared statement.

b) If <statement name> is specified for the <SQL statement name>, then:

i) If <statement name> is not associated with a cursor and either *P* is not a <cursor specification> or *P* is a <cursor specification> that conforms to the Format and Syntax Rules of a <dynamic single row select statement>, then an equivalent <statement name> shall be specified for each <execute statement> that is to be associated with this prepared statement.

ii) If *P* is a <cursor specification> and <statement name> is associated with a cursor *C* through a <dynamic declare cursor>, then an association is made between *C* and *P*. The association is preserved until the prepared statement is destroyed.

14) The validity of an <extended statement name> value or a <statement name> that does not identify a held cursor in an SQL-transaction different from the one in which the statement was prepared is implementation-dependent.

Conformance Rules

1) Without Feature B031, "Basic dynamic SQL", conforming SQL language shall not contain any <prepare statement>.

15.7 <deallocate prepared statement>

Function

Deallocate SQL-statements that have been prepared with a <prepare statement>.

Format

```
<deallocate prepared statement> ::=
      DEALLOCATE PREPARE <SQL statement name>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the containing <SQL-client module definition> shall contain a <prepare statement> whose <statement name> is equivalent to the <statement name> of the <deallocate prepared statement>.

Access Rules

None.

General Rules

1) If the <SQL statement name> does not identify a statement prepared in the scope of the <SQL statement name>, then an exception condition is raised: *invalid SQL statement name.*

2) If the value of <SQL statement name> identifies an existing prepared statement that is the <cursor specification> of an open cursor, then an exception condition is raised: *invalid cursor state.*

3) The prepared statement identified by the <SQL statement name> is destroyed. Any cursor that was allocated with an <allocate cursor statement> that is associated with the prepared statement identified by the <SQL statement name> is destroyed. If the value of the <SQL statement name> identifies an existing prepared statement that is a <cursor specification>, then any prepared statements that reference that cursor are destroyed.

Conformance Rules

1) Without Feature B032, "Extended dynamic SQL", conforming SQL language shall contain no <deallocate prepared statement>.

15.8 <describe statement>

Function

Obtain information about the <select list> columns or <dynamic parameter specification>s contained in a prepared statement or about the columns of the result set associated with a cursor.

Format

```
<describe statement> ::=
          <describe input statement>
        | <describe output statement>

<describe input statement> ::=
        DESCRIBE INPUT <SQL statement name> <using descriptor>
            [ <nesting option> ]

<describe output statement> ::=
        DESCRIBE [ OUTPUT ] <described object> <using descriptor>
            [ <nesting option> ]

<nesting option> ::=
          WITH NESTING
        | WITHOUT NESTING

<using descriptor> ::=
        USING [ SQL ] DESCRIPTOR <descriptor name>

<described object> ::=
          <SQL statement name>
        | CURSOR <extended cursor name> STRUCTURE
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the containing <SQL-client module definition> shall contain a <prepare statement> whose <statement name> is equivalent to the <statement name> of the <describe statement>.

2) If <nesting option> is not specified, then WITHOUT NESTING is implicit.

Access Rules

None.

General Rules

1) If <describe input statement> is executed and the value of the <SQL statement name> does not identify a statement prepared in the scope of the <SQL statement name>, then an exception condition is *invalid SQL statement name*.

2) If <describe output statement> is executed, <SQL statement name> is specified, and the value of the <SQL statement name> does not identify a statement prepared in the scope of the <SQL statement name>, then an exception condition is *invalid SQL statement name*.

3) If <describe output statement> is executed, <extended cursor name> is specified, and the value of the <extended cursor name> does not identify a known allocated cursor, then an exception condition is *invalid cursor name*.

4) An SQL system descriptor area shall have been allocated and not yet deallocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is that specified by the <scope option>. Otherwise, an exception condition is raised: *invalid SQL descriptor name*.

5) Let DA be the descriptor area identified by <descriptor name>. Let N be the <occurrences> specified when DA was allocated.

6) Case:

   a) If the statement being executed is a <describe input statement>, then a descriptor for the input <dynamic parameter specification>s for the prepared statement is stored in DA. Let D be the number of input <dynamic parameter specification>s in the prepared statement. If WITH NESTING is specified, then let NS_i, 1 (one) $\leq i \leq D$, be the number of subordinate descriptors of the descriptor for the i-th input dynamic parameter; otherwise, let NS_i be 0 (zero).

   b) If the statement being executed is a <describe output statement> and the prepared statement that is being described is a <dynamic select statement> or a <dynamic single row select statement>, then a descriptor for the <select list> columns for the prepared statement is stored in DA. Let T be the table defined by the prepared statement and let D be the degree of T. If WITH NESTING is specified, then let NS_i, 1 (one) $\leq i \leq D$, be the number of subordinate descriptors of the descriptor for the i-th column of T; otherwise, let NS_i be 0 (zero).

   c) Otherwise, a descriptor for the output <dynamic parameter specification>s for the prepared statement is stored in DA. Let D be the number of output <dynamic parameter specification>s in the prepared statement. If WITH NESTING is specified, then let NS_i, 1 (one) $\leq i \leq D$, be the number of subordinate descriptors of the descriptor for the i-th output dynamic parameter; otherwise, let NS_i be 0 (zero).

7) DA is set as follows:

   a) Let TD be the value of $D+NS_1+NS_2+ \ldots +NS_D$. COUNT is set to TD.

   b) TOP_LEVEL_COUNT is set to D.

   c) DYNAMIC_FUNCTION and DYNAMIC_FUNCTION_CODE are set to the identifier and code, respectively, for the prepared statement as shown in Table 9, "SQL-statement codes".

   d) If the statement being executed is a <describe output statement> and the prepared statement that is being described is a <dynamic select statement> or a <dynamic single row select statement>:

      Case:

      i) If some subset of the columns of T is the primary key of T, then KEY_TYPE is set to 1 (one).

ii) If some subset of the columns of T is the preferred candidate key of T, then KEY_TYPE is set to 2.

NOTE 12 – Primary keys and preferred candidate keys are defined in Subclause 4.18, "Functional dependencies", ISO/IEC 9075-2.

e) If TD is greater than N, then a completion condition is raised: *warning — insufficient item descriptor areas*.

f) If TD is 0 (zero) or TD is greater than N, then no item descriptor areas are set. Otherwise:

i) The first TD item descriptor areas are set with values from the descriptors and, optionally, subordinate descriptors for

Case:

1) If the statement being executed is a <describe input statement>, then the input <dynamic parameter specification>s.

2) If the statement being executed is a <describe output statement> and the statement being described is a <dynamic select statement> or a <dynamic single row select statement>, then the columns of T.

3) Otherwise, the output <dynamic parameter specification>s.

ii) The descriptor for the first such column or <dynamic parameter specification> is assigned to the first item descriptor area.

iii) If the descriptor for the j-th column or <dynamic parameter specification> is assigned to the k-th item descriptor area, then:

1) The descriptor for the (j+1)-th column or <dynamic parameter specification> is assigned to the (k+NS_{j+1})-th item descriptor area.

2) If WITH NESTING is specified, then the implicitly ordered subordinate descriptors for the j-th column or <dynamic parameter specification> are assigned to contiguous item descriptor areas starting at the (k+1)-th item descriptor area.

8) An SQL item descriptor area, if set, consists of values for LEVEL, TYPE, NULLABLE, NAME, UNNAMED, PARAMETER_ORDINAL_POSITION, PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_SCHEMA, PARAMETER_SPECIFIC_NAME, and other fields depending on the value of TYPE as described below. The DATA and INDICATOR fields are not relevant. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values.

a) If the SQL item descriptor area is set to a descriptor that is immediately subordinate to another whose LEVEL value is K, then LEVEL is set to K+1; otherwise, LEVEL is set to 0 (zero).

b) TYPE is set to a code, as shown in Table 4, "Codes used for SQL data types in Dynamic SQL", indicating the declared type of the column, <dynamic parameter specification>, or subordinate descriptor.

c) Case:

   i) If the value of LEVEL is 0 (zero) and the item descriptor area describes a column, then:

      1) If the column is possibly nullable, then NULLABLE is set to 1 (one); otherwise, NULLABLE is set to 0 (zero).

      2) If the column name is implementation-dependent, then NAME is set to the implementation-dependent name of the column and UNNAMED is set to 1 (one); otherwise, NAME is set to the <derived column> name for the column and UNNAMED is set to 0 (zero).

      3) If the column is a member of the primary key of *T* and KEY_TYPE was set to 1 (one) or if the column is a member of the preferred candidate key of *T* and KEY_TYPE was set to 2, then KEY_MEMBER is set to 1 (one); otherwise, KEY_MEMBER is set to 0 (zero).

   ii) If the value of LEVEL is 0 (zero) and the item descriptor area describes a <dynamic parameter specification>, then:

      1) NULLABLE is set to 1 (one).

         NOTE 13 – This indicates that the <dynamic parameter specification> can have the null value.

      2) UNNAMED is set to 1 (one) and NAME is set to an implementation-dependent value.

      3) KEY_MEMBER is set to 0 (zero).

   iii) Otherwise:

      1) NULLABLE is set to 1 (one).

      2) Case:

         A) If the item descriptor area describes a field of a row, then

            Case:

            I) If the name of the field is implementation-dependent, then NAME is set to the implementation-dependent name of the field and UNNAMED is set to 1 (one).

            II) Otherwise, NAME is set to the name of the field and UNNAMED is set to 0 (zero).

         B) Otherwise, UNNAMED is set to 1 (one) and NAME is set to an implementation-defined value.

      3) KEY_MEMBER is set to 0 (zero).

d) Case:

   i) If TYPE indicates a <character string type>, then LENGTH is set to the length or maximum length in characters of the character string; OCTET_LENGTH is set to the maximum possible length in octets of the character string; CHARACTER_SET_

CATALOG, CHARACTER_SET_SCHEMA and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; and COLLATION_CATALOG, COLLATION_SCHEMA and COLLATION_NAME are set to the <collation name> of the character string's collation. If the subject <language clause> specifies C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string.

ii) If TYPE indicates a <bit string type>, then LENGTH is set to the length or maximum length in bits of the bit string and OCTET_LENGTH is set to the maximum possible length in octets of the bit string.

iii) If TYPE indicates a <binary large object string type>, then LENGTH is set to the length or maximum length in octets of the binary string and OCTET_LENGTH is set to the maximum possible length in octets of the binary string.

iv) If TYPE indicates an <exact numeric type>, then PRECISION and SCALE are set to the precision and scale of the exact numeric.

v) If TYPE indicates an <approximate numeric type>, then PRECISION is set to the precision of the approximate numeric.

vi) If TYPE indicates a <datetime type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 5, "Codes associated with datetime data types in Dynamic SQL", to indicate the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision>, if either is applicable.

vii) If TYPE indicates an <interval type>, then LENGTH is set to the length in positions of the interval type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 6, "Codes used for <interval qualifier>s in Dynamic SQL", to indicate the <interval qualifier> of the interval data type, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision> and PRECISION is set to the <interval fractional seconds precision>, if applicable.

viii) If TYPE indicates a user-defined type, then USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME are set to the <user-defined type name> of the user-defined type's name.

ix) If TYPE indicates a <reference type>, then USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME, SCOPE_CATALOG, SCOPE_SCHEMA, and SCOPE_NAME are set to the <user-defined type name> of the referenced type and qualified name of the referenceable base table; LENGTH and OCTET_LENGTH are set to the implementation-defined length in octets of a <reference type>.

x) If TYPE indicates ROW, then DEGREE is set to the degree of the row type.

xi) If TYPE indicates ARRAY, then CARDINALITY is set to the cardinality of the array type.

e) If LEVEL is 0 (zero) and the prepared statement is a <call statement>, then:

i) Let *SR* be the subject routine for the <routine invocation> of the <call statement>.

ii) Let D_x be the x-th <dynamic parameter specification> simply contained in an SQL argument A_y of the <call statement>.

iii) Let P_y be the y-th SQL parameter of SR.

NOTE 14 – A P whose <SQL parameter mode> is IN can be a <value expression> that contains zero, one, or more <dynamic parameter specification>s. Thus:

— Every D_x maps to one and only one P_y.

— Several D_x instances can map to the same P_y.

— There can be P_y instances that have no D_x instances that map to them.

iv) The PARAMETER_MODE value in the descriptor for each D_x is set to the value from Table 7, "Codes used for input/output SQL parameter modes in Dynamic SQL", that indicates the <SQL parameter mode> of P_y.

v) The PARAMETER_ORDINAL_POSITION value in the descriptor for each D_x is set to the ordinal position of P_y.

vi) The PARAMETER_SPECIFIC_CATALOG, PARAMETER_SPECIFIC_SCHEMA, and PARAMETER_SPECIFIC_NAME values in the descriptor for each D_x are set to the values that identify the catalog, schema, and specific name of SR.

Conformance Rules

1) Without Feature B032, "Extended dynamic SQL", conforming SQL language shall not contain any <describe input statement>.

2) Without Feature B031, "Basic dynamic SQL", conforming SQL language shall not contain any <describe output statement>.

15.9 <input using clause>

Function

Supply input values for an <SQL dynamic statement>.

Format

```
<input using clause> ::=
         <using arguments>
       | <using input descriptor>

<using arguments> ::=
       USING <using argument> [ { <comma> <using argument> }... ]

<using argument> ::= <general value specification>

<using input descriptor> ::= <using descriptor>
```

Syntax Rules

1) The <general value specification> immediately contained in <using argument> shall be either a <host parameter specification> or an <embedded variable specification>.

Access Rules

None.

General Rules

1) If <using input descriptor> is specified, then an SQL system descriptor area shall have been allocated and not yet deallocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is that specified by the <scope option>. Otherwise, an exception condition is raised: *invalid SQL descriptor name*.

2) When an <input using clause> is used in a <dynamic open statement> or as the <parameter using clause> in an <execute statement>, the <input using clause> describes the input <dynamic parameter specification> values for the <dynamic open statement> or the <execute statement>, respectively. Let *PS* be the prepared <dynamic select statement> referenced by the <dynamic open statement> or the prepared statement referenced by the <execute statement>, respectively.

3) Let *D* be the number of input <dynamic parameter specification>s in *PS*.

4) If <using arguments> is specified and the number of <using argument>s is not *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

5) If <using input descriptor> is specified, then:

   a) Let *N* be the value of COUNT.

b) If *N* is greater than the value of <occurrences> specified when the SQL descriptor area identified by <descriptor name> was allocated or is less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.

c) If the first *N* item descriptor areas are not valid as specified in Subclause 15.1, "Description of SQL descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

d) In the first *N* item descriptor areas:

i) If the number of item descriptor areas in which the value of LEVEL is 0 (zero) is not *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

ii) If the value of INDICATOR is not negative, TYPE does not indicate ROW, and the item descriptor area is not subordinate to an item descriptor area whose INDICATOR value is negative or whose TYPE field indicates ARRAY or ARRAY LOCATOR, and if the value of DATA is not a valid value of the data type represented by the item descriptor area, then an exception condition is raised: *dynamic SQL error — using clause does not match dynamic parameter specifications*.

6) For 1 (one) $\leq i \leq D$:

a) Let *TDT* be the effective declared type of the *i*-th input <dynamic parameter specification>, defined to be the type represented by the item descriptor area and its subordinate descriptor areas that would be set by a <describe input statement> to reflect the description of the *i*-th input <dynamic parameter specification> of *PS*.

NOTE 15 – See the General Rules of Subclause 15.8, "<describe statement>".

NOTE 16 – "Represented by", as applied to the relationship between a data type and an item descriptor area, is defined in the Syntax Rules of Subclause 15.1, "Description of SQL descriptor areas".

b) Case:

i) If <using input descriptor> is specified, then:

1) Let *IDA* be the *i*-th item descriptor area whose LEVEL value is 0 (zero).

2) Let *SDT* be the effective declared type represented by *IDA*.

NOTE 17 – "Represented by", as applied to the relationship between a data type and an item descriptor area, is defined in the Syntax Rules of Subclause 15.1, "Description of SQL descriptor areas".

3) Let *SV* be the *associated value* of *IDA*, which is defined to be

Case:

A) If the value of INDICATOR is negative, then *SV* is the null value.

B) Otherwise,

Case:

I) If TYPE indicates ROW, then *SV* is a row whose type is *SDT* and whose field values are the associated values of the immediately subordinate descriptor areas of *IDA*.

II) Otherwise, *SV* is the value of DATA with data type *SDT*.

ii) If <using arguments> is specified, then let *SDT* and *SV* be the declared type and value, respectively, of the *i*-th <using argument>.

c) Case:

i) If *SDT* is a locator type, then:

1) If *SV* is not the null value, then let the value of the *i*-th dynamic parameter be the value of *SV*.

2) Otherwise, let the value of the *i*-th dynamic parameter be the null value.

ii) If *SDT* and *TDT* are predefined data types, then

Case:

1) If the <cast specification>

```
CAST ( IV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2, and there is an implementation-defined conversion from type *STD* to type *TDT*, then that implementation-defined conversion is effectively performed, converting *IV* to type *TDT*, and the result is the value *TV* of the *i*-th input dynamic parameter.

2) Otherwise:

A) If the <cast specification>

```
CAST ( IV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

B) If the <cast specification>

```
CAST ( IV AS TDT )
```

does not conform to the General Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2, then an exception condition is raised in accordance with the General Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2.

C) The <cast specification>

```
CAST ( IV AS TDT )
```

is effectively performed and is the value of the *i*-th input dynamic parameter.

iii) If *SDT* is a predefined data type and *TDT* is a user-defined type, then:

1) Let *DT* be the data type identified by *TDT*.

2) If the current SQL-session has a group name corresponding to the user-defined name of *DT*, then let *GN* be that group name; Otherwise, let *GN* be the default transform group name associated with the current SQL-session.

3) The Syntax Rules of Subclause 10.17, "Determination of a to-sql function", in ISO/IEC 9075-2, are applied with *DT* and *GN* as *TYPE* and *GROUP*, respectively.

   Case:

   A) If there is an applicable to-sql function, then let *TSF* be that to-sql function. If *TSF* is an SQL-invoked method, then let *TSFPT* be the declared type of the second SQL parameter of *TSF*; otherwise, let *TSFPT* be the declared type of the first SQL parameter of *TSF*.

      Case:

      I) If *TSFPT* is compatible with *SDT*, then

         Case:

         1) If *TSF* is an SQL-invoked method, then *TSF* is effectively invoked with the value returned by the function invocation:

            ```
            DT( )
            ```

            as the first parameter and *SV* as the second parameter. The <return value> is the value of the *i*-th input dynamic parameter.

         2) Otherwise, *TSF* is effectively invoked with *SV* as the first parameter. The <return value> is the value of the *i*-th input dynamic parameter.

      II) Otherwise, an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

   B) Otherwise, an exception condition is raised: *dynamic SQL error — data type transform function violation*.

Conformance Rules

1) Without Feature B032, “Extended dynamic SQL”, a <descriptor name> shall be a <literal>.

2) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <using input clause>.

15.10 <output using clause>

Function

Supply output variables for an <SQL dynamic statement>.

Format

```
<output using clause> ::=
          <into arguments>
        | <into descriptor>

<into arguments> ::=
        INTO <into argument> [ { <comma> <into argument> }... ]

<into argument> ::= <target specification>

<into descriptor> ::=
        INTO [ SQL ] DESCRIPTOR <descriptor name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If <into descriptor> is specified, then an SQL system descriptor area shall have been allocated and not yet deallocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is that specified by the <scope option>. Otherwise, an exception condition is raised: *invalid SQL descriptor name.*

2) When an <output using clause> is used in a <dynamic fetch statement> or as the <result using clause> of an <execute statement>, let *PS* be the prepared <dynamic select statement> referenced by the <dynamic fetch statement> or the prepared <dynamic single row select statement> referenced by the <execute statement>, respectively.

3) Case:

   a) If *PS* is a <prepared dynamic select statement> or a <dynamic single row select statement>, then the <output using clause> describes the <target specification>s for the <dynamic fetch statement> or the <execute statement>. Let *D* be the degree of the table specified by *PS*.

   b) Otherwise, the <output using clause> describes the <target specification>s for the output <dynamic parameter specification>s contained in *PS*. Let *D* be the number of such output <dynamic parameter specification>s.

4) If <into arguments> is specified and the number of <into argument>s is not *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications.*

5) If <into descriptor> is specified, then:

   a) Let *N* be the value of COUNT.

   b) If *N* is greater than the value of <occurrences> specified when the SQL descriptor area identified by <descriptor name> was allocated or less than zero, then an exception condition is raised: *dynamic SQL error — invalid descriptor count*.

   c) If the first *N* item descriptor areas are not valid as specified in Subclause 15.1, "Description of SQL descriptor areas", then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

   d) In the first *N* item descriptor areas, if the number of item descriptor areas in which the value of LEVEL is 0 (zero) is not *D*, then an exception condition is raised: *dynamic SQL error — using clause does not match target specifications*.

6) For 1 (one) ≤ *i* ≤ *D*:

   a) Let *SDT* be the effective declared type of the *i*-th <select list> column or output <dynamic parameter specification>, defined to be the type represented by the item descriptor area and its subordinate descriptor areas that would be set by

      Case:

      i) If *PS* is a <prepared dynamic select statement> or a <dynamic single row select statement>, then a <describe output statement> to reflect the description of the *i*-th <select list> column; let *SV* be the value of that <select list> column, with data type *SDT*.

      ii) Otherwise, a <describe output statement> to reflect the description of the *i*-th output <dynamic parameter specification>; let *SV* be the value of that <dynamic parameter specification>, with data type *SDT*.

      NOTE 18 – "Represented by", as applied to the relationship between a data type and an item descriptor area, is defined in the Syntax Rules of Subclause 15.1, "Description of SQL descriptor areas".

   b) Case:

      i) If <into descriptor> is specified, then let *TDT* be the declared type of the *i*-th <target specification> as represented by the *i*-th item descriptor area *IDA* whose LEVEL value is 0 (zero).

         NOTE 19 – "Represented by", as applied to the relationship between a data type and an item descriptor area, is defined in the Syntax Rules of Subclause 15.1, "Description of SQL descriptor areas".

      ii) If <into arguments> is specified, then let *TDT* be the data type of the *i*-th <into argument>.

   c) If the <output using clause> is used in a <dynamic fetch statement>, then let *LTDT* be the data type on the most recently executed <dynamic fetch statement>, if any, for the cursor *CR*. It is implementation-defined whether or not an exception condition is raised: *dynamic SQL error — restricted data type attribute violation* if any of the following are true:

      i) *LTDT* and *TDT* both identify a binary large object type and only one of *LTDT* and *TDT* is a binary large object locator.

ii) *LTDT* and *TDT* both identify a character large object type and only one of *LTDT* and *TDT* is a character large object locator.

iii) *LTDT* and *TDT* both identify an array type and only one of *LTDT* and *TDT* is an array locator.

iv) *LTDT* and *TDT* both identify a user-defined type and only one of *LTDT* and *TDT* is a user-defined type locator.

d) Case:

i) If *TDT* is a locator type, then:

1) If *SV* is not the null value, then a locator *L* that uniquely identifies *SV* is generated and is the value *TV* of the *i*-th <target specification>.

2) Otherwise, the value *TV* of the *i*-th <target specification> is the null value.

ii) If *STD* and *TDT* are predefined data types, then

Case:

1) If the <cast specification>

```
CAST ( SV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2, and there is an implementation-defined conversion of type *STD* to type *TDT*, then that implementation-defined conversion is effectively performed, converting *SV* to type *TDT*, and the result is the value *TV* of the *i*-th <target specification>.

2) Otherwise:

A) If the <cast specification>

```
CAST ( SV AS TDT )
```

does not conform to the Syntax Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2, then an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

B) If the <cast specification>

```
CAST ( SV AS TDT )
```

does not conform to the General Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2, then an exception condition is raised in accordance with the General Rules of Subclause 6.22, "<cast specification>", in ISO/IEC 9075-2.

C) The <cast specification>

```
CAST ( SV AS TDT )
```

is effectively performed, and is the value *TV* of the *i*-th <target specification>.

iii) If *SDT* is a user-defined type and *TDT* is a predefined data type, then:

1) Let *DT* be the data type identified by *SDT*.

2) If the current SQL-session has a group name corresponding to the user-defined type name of *DT*, then let *GN* be that group name; otherwise, let *GN* be the default transform group name associated with the current SQL-session.

3) Apply the Syntax Rules of Subclause 10.15, "Determination of a from-sql function", in ISO/IEC 9075-2, with *DT* and *GN* as *TYPE* and *GROUP*, respectively.

Case:

A) If there is an applicable from-sql function, then let *FSF* be that from-sql function and let *FSFRT* be the <returns data type> of *FSF*.

Case:

I) If *FSFRT* is compatible with *TDT*, then the from-sql function *FSF* is effectively invoked with *SV* as its input parameter and the <return value> is the value *TV* of the *i*-th <target specification>.

II) Otherwise, an exception condition is raised: *dynamic SQL error — restricted data type attribute violation*.

B) Otherwise, an exception condition is raised: *dynamic SQL error — data type transform function violation*.

e) Case:

i) If <into descriptor> is specified, then *IDA* is set to reflect the value of *TV* as follows:

Case:

1) If TYPE indicates ROW, then

Case:

A) If *TV* is the null value, then the value of INDICATOR in *IDA* and in all subordinate descriptor areas of *IDA* that are not subordinate to an item descriptor area whose TYPE indicates ARRAY or ARRAY LOCATOR is set to −1.

B) Otherwise, the *i*-th subordinate descriptor area of *IDA* is set to reflect the value of the *i*-th field of *TV* by applying this subrule (beginning with the outermost 'Case') to the *i*-th subordinate descriptor area of *IDA* as *IDA*, the value of the *i*-th field of *TV* as *TV*, the value of the *i*-th field of *SV* as *SV*, and the data type of the *i*-th field of *SV* as *SDT*.

2) Otherwise,

Case:

A) If *TV* is the null value, then the value of INDICATOR is set to −1.

B) If *TV* is not the null value, then:

I) The value of INDICATOR is set to 0 (zero).

II) Case:

1) If TYPE indicates a locator type, then a locator *L* that uniquely identifies *TV* is generated and the value of DATA is set to an implementation-dependent four-octet value that represents *L*.

2) Otherwise, the value of DATA is set to *TV*.

III) Case:

1) If TYPE indicates CHARACTER VARYING, BIT VARYING, or BINARY LARGE OBJECT, then RETURNED_LENGTH is set to the length in characters, bits, or octets, respectively, of *TV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *TV*.

2) If *SDT* is CHARACTER VARYING, BIT VARYING, or BINARY LARGE OBJECT, then RETURNED_LENGTH is set to the length in characters, bits, or octets, respectively, of *SV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *SV*.

3) If TYPE indicates ARRAY or ARRAY LOCATOR, then RETURNED_CARDINALITY is set to the cardinality of *TV*.

ii) If <into arguments> is specified, then the Rules in Subclause 9.1, "Retrieval assignment", are applied to *TV* and the *i*-th <into argument> as *VALUE* and *TARGET*, respectively.

NOTE 20 – All other values of the SQL descriptor area are unchanged.

Conformance Rules

1) Without Feature B032, "Extended dynamic SQL", a <descriptor name> shall be a <literal>.

2) Without Feature B031, "Basic dynamic SQL", conforming SQL language shall not contain any <output using clause>.

15.11 <execute statement>

Function

Associate input SQL parameters and output targets with a prepared statement and execute the statement.

Format

```
<execute statement> ::=
       EXECUTE <SQL statement name>
         [ <result using clause> ]
         [ <parameter using clause> ]

<result using clause> ::= <output using clause>

<parameter using clause> ::= <input using clause>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the containing <SQL-client module definition> shall contain a <prepare statement> whose <statement name> is equivalent to the <statement name> of the <execute statement>.

Access Rules

None.

General Rules

1) When the <execute statement> is executed, if the <SQL statement name> does not identify a statement *P* previously prepared in the scope of the <SQL statement name>, then an exception condition is raised: *invalid SQL statement name*.

2) Let *PS* be the statement previously prepared using <SQL statement name>.

3) If *PS* is a <dynamic select statement> that does not conform to the Format and Syntax Rules of a <dynamic single row select statement>, then an exception condition is raised: *dynamic SQL error — cursor specification cannot be executed*.

4) If *PS* contains the <table name> of a created or declared local temporary table and if the <execute statement> is not in the same <SQL-client module definition> as the <prepare statement> that prepared the prepared statement, then an exception condition is raised: *syntax error or access rule violation*.

5) If *PS* contains input <dynamic parameter specification>s and a <parameter using clause> is not specified, then an exception condition is raised: *dynamic SQL error — using clause required for dynamic parameters*.

6) If *PS* is a <dynamic single row select statement> or it contains output <dynamic parameter specification>s and a <result using clause> is not specified, then an exception condition is raised: *dynamic SQL error — using clause required for result fields*.

7) If a <parameter using clause> is specified, then the General Rules specified in Subclause 15.9, “<input using clause>”, for a <parameter using clause> in an <execute statement> are applied.

8) *PS* is executed by applying the General Rules of Subclause 11.3, “<SQL procedure statement>”, to *PS* as if it were immediately contained in an <externally-invoked procedure>. All General Rules are applied except those associated with input and output parameters.

9) If *PA* is a <preparable dynamic delete statement: positioned>, then when it is executed all General Rules in Subclause 15.21, “<preparable dynamic delete statement: positioned>”, apply to the <preparable statement>.

10) If *PS* is a <preparable dynamic update statement: positioned>, then when it is executed, all General Rules in Subclause 15.22, “<preparable dynamic update statement: positioned>”, apply to the <preparable statement>.

11) If a <result using clause> is specified, then the General Rules specified in Subclause 15.10, “<output using clause>”, for a <result using clause> in an <execute statement> are applied.

Conformance Rules

1) Without Feature B032, “Extended dynamic SQL”, conforming SQL language shall contain no <result using clause>.

2) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <execute statement>.

15.12 <execute immediate statement>

Function

Dynamically prepare and execute a preparable statement.

Format

```
<execute immediate statement> ::=
      EXECUTE IMMEDIATE <SQL statement variable>
```

Syntax Rules

1) The declared type of <SQL statement variable> shall be character string.

Access Rules

None.

General Rules

1) Let *P* be the contents of the <SQL statement variable>.

2) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation*.

   a) *P* is a <dynamic select statement> or a <dynamic single row select statement>.

   b) *P* contains a <dynamic parameter specification>.

3) Let *SV* be <SQL statement variable>. <execute immediate statement> is equivalent to the following:

```
PREPARE IMMEDIATE_STMT FROM SV ;
EXECUTE IMMEDIATE_STMT ;
DEALLOCATE PREPARE IMMEDIATE_STMT ;
```

   where *IMMEDIATE_STMT* is an implementation-defined <statement name> that is not equivalent to any other <statement name> in the containing <SQL-client module definition>.

   NOTE 21 – Exception condition or completion condition information resulting from the PREPARE or EXECUTE is reflected in the diagnostics area.

Conformance Rules

1) Without Feature B031, "Basic dynamic SQL", conforming SQL language shall not contain any <execute immediate statement>.

15.13 <dynamic declare cursor>

Function

Declare a cursor to be associated with a <statement name>, which may in turn be associated with a <cursor specification>.

Format

```
<dynamic declare cursor> ::=
       DECLARE <cursor name> [ <cursor sensitivity>  ]
         [ <cursor scrollability> ] CURSOR
         [ <cursor holdability> ]
         [ <cursor returnability> ]
           FOR <statement name>
```

Syntax Rules

1) The <cursor name> shall not be identical to the <cursor name> specified in any other <declare cursor> or <dynamic declare cursor> in the same <SQL-client module definition>.

2) The containing <SQL-client module definition> shall contain a <prepare statement> whose <statement name> is equivalent to the <statement name> of the <dynamic declare cursor>.

3) If <cursor scrollability is not specified, then NO SCROLL is implicit.

4) If <cursor holdability is not specified, then WITHOUT HOLD is implicit.

5) If <cursor returnability is not specified, then WITHOUT RETURN is implicit.

Access Rules

None.

General Rules

1) All General Rules of Subclause 14.1, "<declare cursor>", in ISO/IEC 9075-2, apply to <dynamic declare cursor>, replacing “<cursor specification>” with “prepared statement”.

Conformance Rules

1) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <dynamic declare cursor>.

2) Without Feature B031, “Basic dynamic SQL”, and Feature T231, “SENSITIVE cursors”, a <dynamic declare cursor> shall not specify SENSITIVE.

3) Without Feature B031, “Basic dynamic SQL”, and Feature F791, “Insensitive cursors”, a <dynamic declare cursor> shall not specify INSENSITIVE.

4) Without Feature B031, “Basic dynamic SQL”, and either Feature F791, “Insensitive cursors”, or Feature T231, “SENSITIVE cursors”, a <dynamic declare cursor> shall not specify ASENSITIVE.

5) Without Feature B031, “Basic dynamic SQL”, and Feature T471, “Result sets return value”, a <dynamic declare cursor> shall not specify <cursor returnability>.

6) Without Feature B031, “Basic dynamic SQL”, and Feature F341, “Scrolled cursors”, and Feature F831, “Full cursor update”, if an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then <cursor scrollability> shall not be specified.

7) Without Feature B031, “Basic dynamic SQL”, and Feature F831, “Full cursor update”, if an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then ORDER BY shall not be specified.

8) Without Feature B031, “Basic dynamic SQL”, and Feature F421, “Scrolled cursors”, a <dynamic declare cursor> shall not specify <cursor scrollability>.

9) Without Feature T551, “Optional key words for default syntax”, a <dynamic declare cursor> shall not specify WITHOUT HOLD.

10) Without Feature S024, “Enhanced structured types”, a <value expression> that is a <sort key> shall not be of a structured type.

15.14 <allocate cursor statement>

Function

Define a cursor based on a prepared statement for a <cursor specification> or assign a cursor to the ordered set of result sets returned from an SQL-invoked procedure.

Format

```
<allocate cursor statement> ::=
      ALLOCATE <extended cursor name> <cursor intent>

<cursor intent> ::=
        <statement cursor>
      | <result set cursor>

<statement cursor> ::=
      [ <cursor sensitivity> ] [ SCROLL ] CURSOR
        [ WITH HOLD ]
        [ WITH RETURN ]
          FOR <extended statement name>

<result set cursor> ::=
      FOR PROCEDURE <specific routine designator>
```

Syntax Rules

1) If <result set cursor> is specified, then the SQL-invoked routine identified by <specific routine designator> shall be an SQL-invoked procedure.

Access Rules

None.

General Rules

1) Let *S* be the <simple value specification> immediately contained in <extended cursor name>. Let *V* be the character string that is the result of

   `TRIM ( BOTH ' ' FROM S )`

   If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid cursor name.*

2) If the value of the <extended cursor name> is identical to the value of the <extended cursor name> of any other cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name.*

3) If <statement cursor> is specified, then:

   a) When the <allocate cursor statement> is executed, if the value of the <extended statement name> does not identify a statement previously prepared in the scope of the <extended statement name>, then an exception condition is raised: *invalid SQL statement name.*

b) If the prepared statement associated with the <extended statement name> is not a <cursor specification>, then an exception condition is raised: *dynamic SQL error — prepared statement not a cursor specification*.

c) All General Rules of Subclause 14.1, "<declare cursor>", in ISO/IEC 9075-2, apply to <allocate cursor statement>, replacing “<open statement>” with “<dynamic open statement>” and “<cursor specification>” with “prepared statement”.

d) An association is made between the value of the <extended cursor name> and the prepared statement in the scope of the <extended cursor name>. The association is preserved until the prepared statement is destroyed, at which time the cursor identified by <extended cursor name> is also destroyed.

4) If <result set cursor> is specified, then:

a) When the <allocate cursor statement> is executed, if the <specific routine designator> does not identify an SQL-invoked procedure *P* that has been previously invoked during the current SQL-session, an exception condition is raised: *invalid SQL-invoked procedure reference*.

b) If *P* did not return any result sets, then an exception condition is raised: *no data — no additional dynamic result sets returned*.

c) Let *RRS* be the ordered set of result sets returned by *P*.

d) When the <allocate cursor statement> is executed, an association is made between the <extended cursor name> and the first result set *FRS* in *RRS*. The definition of *FRS* is the definition of the <cursor specification> *CS* in *P* that created *FRS*. Let *CR* be the cursor declared by the <declare cursor> that contains *CS*.

e) Let *T* be the table specified by *CS*. *T* is the first result set returned from *P*.

f) A table descriptor for *T* is effectively created.

g) Cursor *CR* is placed in the open state and its position is before the first row of *T*.

Conformance Rules

1) Without Feature B032, “Extended dynamic SQL”, conforming SQL language shall not contain any <allocate cursor statement>.

15.15 <dynamic open statement>

Function

Associate input dynamic parameters with a <cursor specification> and open the cursor.

Format

```
<dynamic open statement> ::=
      OPEN <dynamic cursor name> [ <input using clause> ]
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <SQL-client module definition> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

Access Rules

1) The Access Rules for the <query expression> simply contained in the prepared statement associated with the <dynamic cursor name> are applied.

General Rules

1) If <dynamic cursor name> is a <cursor name> and the <statement name> of the associated <dynamic declare cursor> is not associated with a prepared statement, then an exception condition is raised: *invalid SQL statement name.*

2) If <dynamic cursor name> is an <extended cursor name> whose value does not identify a cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name.*

3) If the prepared statement associated with the <dynamic cursor name> contains <dynamic parameter specification>s and an <input using clause> is not specified, then an exception condition is raised: *dynamic SQL error — using clause required for dynamic parameters.*

4) The cursor specified by <dynamic cursor name> is updatable if and only if the associated <cursor specification> specified an updatable cursor.

   NOTE 22 – “updatable cursor” is defined in Subclause 14.1, "<declare cursor>", in ISO/IEC 9075-2.

5) If an <input using clause> is specified, then the General Rules specified in Subclause 15.9, “<input using clause>”, for <dynamic open statement> are applied.

6) All General Rules of Subclause 14.2, "<open statement>", in ISO/IEC 9075-2, apply to the <dynamic open statement>.

Conformance Rules

1) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <dynamic open statement>.

15.16 <dynamic fetch statement>

Function

Fetch a row for a cursor declared with a <dynamic declare cursor>.

Format

```
<dynamic fetch statement> ::=
      FETCH [ [ <fetch orientation> ] FROM ] <dynamic cursor name> <output using clause>
```

Syntax Rules

1) If <fetch orientation> is omitted, then NEXT is implicit.

2) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <SQL-client module definition> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

3) Let *CR* be the cursor identified by *DCN*.

4) If the implicit or explicit <fetch orientation> is not NEXT, then the <dynamic declare cursor> or <allocate cursor statement> associated with *CR* shall specify SCROLL.

Access Rules

None.

General Rules

1) All General Rules of Subclause 14.3, "<fetch statement>", in ISO/IEC 9075-2 are applied to cursor *CR*, <fetch orientation>, and an empty <fetch target list>.

2) The General Rules of Subclause 15.10, "<output using clause>", are applied to the <dynamic fetch statement> and the current row of *CR* as the retrieved row.

Conformance Rules

1) Without Feature B031, "Basic dynamic SQL", conforming SQL language shall not contain any <dynamic fetch statement>.

15.17 <dynamic single row select statement>

Function

Retrieve values from a dynamically-specified row of a table.

Format

```
<dynamic single row select statement> ::= <query specification>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *Q* be the result of the <query specification>.

2) Case:

   a) If the cardinality of *Q* is greater than 1 (one), then an exception condition is raised: *cardinality violation*.

   b) If *Q* is empty, then a completion condition is raised: *no data*.

Conformance Rules

1) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall contain no <dynamic single row select statement>.

15.18 <dynamic close statement>

Function

Close a cursor.

Format

```
<dynamic close statement> ::=
       CLOSE <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <SQL-client module definition> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

Access Rules

None.

General Rules

1) All General Rules of Subclause 14.4, "<close statement>", in ISO/IEC 9075-2, apply to the <dynamic close statement>.

Conformance Rules

1) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <dynamic close statement>.

15.19 <dynamic delete statement: positioned>

Function

Delete a row of a table.

Format

```
<dynamic delete statement: positioned> ::=
      DELETE FROM <target table>
        WHERE CURRENT OF <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <SQL-client module definition> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *T* be the table identified by the <table name> contained in <target table>.

Access Rules

1) All Access Rules of Subclause 14.6, "<delete statement: positioned>", in ISO/IEC 9075-2, apply to the <dynamic delete statement: positioned>.

General Rules

1) If *DCN* is a <cursor name> and the <statement name> of the associated <dynamic declare cursor> is not associated with a prepared statement, then an exception condition is raised: *invalid SQL statement name*.

2) If *DCN* is an <extended cursor name> whose value does not identify a cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name*.

3) Let *CR* be the cursor identified by *DCN*.

4) If *CR* is not an updatable cursor, then an exception condition is raised: *invalid cursor name*.

5) Let *T* be the simply underlying table of *CR*. *T* is the subject table of the <dynamic delete statement: positioned>. *T* shall have exactly one leaf underlying table *LUT*. Let *LUTN* be a <table name> that identifies *LUT*.

6) Let *TN* be the <table name> contained in <target table>. If *TN* does not identify *LUTN*, or if ONLY is specified and the <table reference> in *T* that references *LUT* does not specify ONLY, or if ONLY is not specified and the <table reference> in *T* that references *LUT* does specify ONLY, then an exception condition is raised: *target table disagrees with cursor specification*.

7) All General Rules of Subclause 14.6, "<delete statement: positioned>", in ISO/IEC 9075-2, apply to the <dynamic delete statement: positioned>, replacing "<delete statement: positioned>" with "<dynamic delete statement: positioned>".

Conformance Rules

1) Without Feature B031, “Basic dynamic SQL”, conforming SQL language shall not contain any <dynamic delete statement: positioned>.

15.20 <dynamic update statement: positioned>

Function

Update a row of a table.

Format

```
<dynamic update statement: positioned> ::=
      UPDATE <target table>
        SET <set clause list>
          WHERE CURRENT OF <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <SQL-client module definition> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *T* be the table identified by the <table name> contained in <target table>.

3) The scope of the <table name> is the entire <dynamic update statement: positioned>.

Access Rules

1) All Access Rules of Subclause 14.9, "<update statement: positioned>", in ISO/IEC 9075-2, apply to the <dynamic update statement: positioned>.

General Rules

1) If *DCN* is a <cursor name> and the <statement name> of the associated <dynamic declare cursor> is not associated with a prepared statement, then an exception condition is raised: *invalid SQL statement name.*

2) If *DCN* is an <extended cursor name> whose value does not identify a cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name.*

3) Let *CR* be the cursor identified by *DCN*.

4) If *CR* is not an updatable cursor, then an exception condition is raised: *invalid cursor name.*

5) Let *T* be the simply underlying table of *CR*. *T* is the subject table of the <dynamic update statement: positioned>. *T* shall have exactly one leaf underlying table *LUT*. Let *LUTN* be a <table name> that identifies *LUT*.

6) Let *TN* be the <table name> contained in <target table>. If *TN* does not identify *LUTN*, or if ONLY is specified and the <table reference> in *T* that references *LUT* does not specify ONLY, or if ONLY is not specified and the <table reference> in *T* that references *LUT* does specify ONLY, then an exception condition is raised: *target table disagrees with cursor specification*.

7) If any object column is directly or indirectly referenced in the <order by clause> of the <cursor specification> for *CR*, then an exception condition is raised: *attempt to assign to ordering column*.

8) If any object column identifies a column that is not identified by a <column name> contained in the explicit or implicit <column name list> of the explicit or implicit <updatability clause> of the <cursor specification> for *CR*, then an exception condition is raised: *attempt to assign to non-updatable column*.

9) All General Rules of Subclause 14.9, "<update statement: positioned>", in ISO/IEC 9075-2, apply to the <dynamic update statement: positioned>, replacing "<cursor name>" with "<dynamic cursor name>" and "<update statement: positioned>" with "<dynamic update statement: positioned>".

Conformance Rules

1) Without Feature B031, "Basic dynamic SQL", conforming SQL language shall not contain any <dynamic update statement: positioned>.

15.21 <preparable dynamic delete statement: positioned>

Function

Delete a row of a table through a dynamic cursor.

Format

```
<preparable dynamic delete statement: positioned> ::=
      DELETE [ FROM <target table> ]
        WHERE CURRENT OF [ <scope option> ] <cursor name>
```

Syntax Rules

1) If <target table> is not specified, then let *TN* be the name of the leaf underlying table *LUT* of the <cursor specification> identified by <cursor name>.

   Case:

   i) If the <table reference> that references *LUT* specifies ONLY, then the <target table>

      ```
      ONLY ( TN )
      ```

      is implicit.

   ii) Otherwise, the <target table>

      ```
      TN
      ```

      is implicit.

2) All Syntax Rules of Subclause 14.6, "<delete statement: positioned>", in ISO/IEC 9075-2, apply to the <preparable dynamic delete statement: positioned>, replacing "<declare cursor>" with "<dynamic declare cursor> or <allocate cursor statement>" and "<delete statement: positioned>" with "<preparable dynamic delete statement: positioned>".

Access Rules

1) All Access Rules of Subclause 14.6, "<delete statement: positioned>", in ISO/IEC 9075-2, apply to the <preparable dynamic delete statement: positioned>.

General Rules

1) All General Rules of Subclause 14.6, "<delete statement: positioned>", in ISO/IEC 9075-2, apply to the <preparable dynamic delete statement: positioned>, replacing "<delete statement: positioned>" with "<preparable dynamic delete statement: positioned>".

Conformance Rules

1) Without Feature B032, "Extended dynamic SQL", conforming SQL language shall contain no <preparable dynamic delete statement: positioned>.

15.22 <preparable dynamic update statement: positioned>

Function

Update a row of a table through a dynamic cursor.

Format

```
<preparable dynamic update statement: positioned> ::=
      UPDATE [ <target table> ]
        SET <set clause list>
        WHERE CURRENT OF [ <scope option> ] <cursor name>
```

Syntax Rules

1) If <target table> is not specified, then let *TN* be the name of the leaf underlying table *LUT* of the <cursor specification> identified by <cursor name>.

   Case:

   i) If the <table reference> that references *LUT* specifies ONLY, then the <target table>

      ```
      ONLY ( TN )
      ```

      is implicit.

   ii) Otherwise, the <target table>

      ```
      TN
      ```

      is implicit.

2) All Syntax Rules of Subclause 14.9, "<update statement: positioned>", in ISO/IEC 9075-2, apply to the <preparable dynamic update statement: positioned>, replacing “<declare cursor>” with “<dynamic declare cursor> or <allocate cursor statement>” and “<update statement: positioned>” with “<preparable dynamic update statement: positioned>”.

Access Rules

1) All Access Rules of Subclause 14.9, "<update statement: positioned>", in ISO/IEC 9075-2, apply to the <preparable dynamic update statement: positioned>.

General Rules

1) All General Rules of Subclause 14.9, "<update statement: positioned>", in ISO/IEC 9075-2, apply to the <preparable dynamic update statement: positioned>, replacing “<update statement: positioned>” with “<preparable dynamic update statement: positioned>”.

Conformance Rules

1) Without Feature B032, “Extended dynamic SQL”, conforming SQL language shall contain no <preparable dynamic update statement: positioned>.

16 Embedded SQL

16.1 <embedded SQL host program>

Function

Specify an <embedded SQL host program>.

Format

```
<embedded SQL host program> ::=
          <embedded SQL Ada program>
        | <embedded SQL C program>
        | <embedded SQL COBOL program>
        | <embedded SQL Fortran program>
        | <embedded SQL MUMPS program>
        | <embedded SQL Pascal program>
        | <embedded SQL PL/I program>

<embedded SQL statement> ::=
        <SQL prefix>
          <statement or declaration>
        [ <SQL terminator> ]

<statement or declaration> ::=
          <declare cursor>
        | <dynamic declare cursor>
        | <temporary table declaration>
        | <embedded authorization declaration>
        | <embedded path specification>
        | <embedded transform group specification>
        | <embedded exception declaration>
        | <handler declaration>
        | <SQL-invoked routine>
        | <SQL procedure statement>

<SQL prefix> ::=
          EXEC SQL
        | <ampersand>SQL<left paren>

<SQL terminator> ::=
          END-EXEC
        | <semicolon>
        | <right paren>

<embedded authorization declaration> ::=
        DECLARE <embedded authorization clause>

<embedded authorization clause> ::=
          SCHEMA <schema name>
        | AUTHORIZATION <embedded authorization identifier>
              [ FOR STATIC { ONLY | AND DYNAMIC } ]
        | SCHEMA <schema name> AUTHORIZATION <embedded authorization identifier>
              [ FOR STATIC { ONLY | AND DYNAMIC } ]
```

```
<embedded authorization identifier> ::=
       <module authorization identifier>
       <embedded path specification> ::=
       <path specification>

<embedded transform group specification> ::=
       <transform group specification>

<embedded SQL declare section> ::=
         <embedded SQL begin declare>
           [ <embedded character set declaration> ]
           [ <host variable definition>... ]
         <embedded SQL end declare>
       | <embedded SQL MUMPS declare>

<embedded character set declaration> ::=
       SQL NAMES ARE <character set specification>

<embedded SQL begin declare> ::=
       <SQL prefix> BEGIN DECLARE SECTION [ <SQL terminator> ]

<embedded SQL end declare> ::=
       <SQL prefix> END DECLARE SECTION [ <SQL terminator> ]

<embedded SQL MUMPS declare> ::=
       <SQL prefix>
         BEGIN DECLARE SECTION
           [ <embedded character set declaration> ]
           [ <host variable definition>... ]
         END DECLARE SECTION
       <SQL terminator>

<host variable definition> ::=
         <Ada variable definition>
       | <C variable definition>
       | <COBOL variable definition>
       | <Fortran variable definition>
       | <MUMPS variable definition>
       | <Pascal variable definition>
       | <PL/I variable definition>

<embedded variable name> ::=
       <colon><host identifier>

<host identifier> ::=
         <Ada host identifier>
       | <C host identifier>
       | <COBOL host identifier>
       | <Fortran host identifier>
       | <MUMPS host identifier>
       | <Pascal host identifier>
       | <PL/I host identifier>
```

Syntax Rules

1) An <embedded SQL host program> is a compilation unit that consists of programming language text and SQL text. The programming language text shall conform to the requirements of a specific standard programming language. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s, as defined in this International Standard.

   NOTE 23 – "Compilation unit" is defined in Subclause 4.21, "SQL-client modules", in ISO/IEC 9075-2.

2) Case:

   a) An <embedded SQL statement> or <embedded SQL MUMPS declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL prefix> that is "<ampersand>SQL<left paren>". There shall be no <separator> between the <ampersand> and "SQL" nor between "SQL" and the <left paren>.

   b) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> that is not contained in an <embedded SQL MUMPS program> shall contain an <SQL prefix> that is "EXEC SQL".

3) Case:

   a) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL COBOL program> shall contain an <SQL terminator> that is END-EXEC.

   b) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL Fortran program> shall not contain an <SQL terminator>.

   c) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL Pascal program>, or <embedded SQL PL/I program> shall contain an <SQL terminator> that is a <semicolon>.

   d) An <embedded SQL statement> or <embedded SQL MUMPS declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL terminator> that is a <right paren>.

4) Case:

   a) An <embedded SQL declare section> that is contained in an <embedded SQL MUMPS program> shall be an <embedded SQL MUMPS declare>.

   b) An <embedded SQL declare section> that is not contained in an <embedded SQL MUMPS program> shall not be an <embedded SQL MUMPS declare>.

   NOTE 24 – There is no restriction on the number of <embedded SQL declare section>s that may be contained in an <embedded SQL host program>.

5) The <token>s comprising an <SQL prefix>, <embedded SQL begin declare>, or <embedded SQL end declare> shall be separated by <space> characters and shall be specified on one line. Otherwise, the rules for the continuation of lines and tokens from one line to the next and for the placement of host language comments are those of the programming language of the containing <embedded SQL host program>.

6) If an <embedded authorization declaration> appears in an <embedded SQL host program>, then it shall be contained in the first <embedded SQL statement> of that <embedded SQL host program>.

7) An <embedded SQL host program> shall not contain more than one <embedded path specification>.

8) An <embedded SQL host program> shall not contain more than one <embedded transform group specification>.

9) Case:

   i) If <embedded transform group specification> is not specified, then an <embedded transform group specification> containing a <multiple group specification> with a <group specification> *GS* for each <host variable declaration> that has an associated user-defined type *UDT*, but is not a user-defined locator variable is implicit. The <group name> of *GS* is implementation-defined and its <user-defined type> is *UDT*.

   ii) If <embedded transform group specification> contains a <single group specification> with a <group name> *GN*, then an <embedded transform group specification> containing a <multiple group specification> with a <group specification> *GS* for each <host variable declaration> that has an associated user-defined type *UDT*, but is not a user-defined type locator variable is implicit. The <group name> of *GS* is *GN* and its <user-defined type> is *UDT*.

   iii) If <embedded transform group specification> contains a <multiple group specification> *MGS*, then an <embedded transform group specification> containing a <multiple group specification> that contains *MGS* extended with a <group specification> *GS* for each <host variable declaration> that has an associated user-defined type *UDT*, but is not a user-defined locator variable and the <user-defined type name> of *UDT* is not contained in any <group specification> contained in *MGS* is implicit. The <group name> of *GS* is implementation-defined and its <user-defined type> is *UDT*.

10) The implicit or explicit <embedded transform group specification> precedes in the text of the <embedded SQL host program> every <host variable declaration>.

11) An <embedded SQL host program> shall contain no more than one <embedded character set declaration>. If an <embedded character set declaration> is not specified, then an <embedded character set declaration> that specifies an implementation-defined character set that contains at least every character that is in <SQL language character> is implicit.

12) A <temporary table declaration> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement or <declare cursor> that references the <table name> of the <temporary table declaration>.

13) A <declare cursor> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <cursor name> of the <declare cursor>.

14) A <dynamic declare cursor> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <cursor name> of the <dynamic declare cursor>.

15) An <embedded exception declaration> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <exception name> contained in the <embedded exception declaration>.

16) An <exception name> contained in an <embedded exception declaration> that is immediately contained in an <embedded SQL host program> shall not be equivalent to any other <exception name> contained in any other <embedded exception declaration> that is immediately contained in the <embedded SQL host program>.

17) Any <host identifier> that is contained in an <embedded SQL statement> in an <embedded SQL host program> shall be defined in exactly one <host variable definition> contained in that <embedded SQL host program>. In programming languages that support <host variable definition>s in subprograms, two <host variable definition>s with different, non-overlapping scope in the host language are to be regarded as defining different host variables, even if they specify the same variable name. That <host variable definition> shall appear in the text of the <embedded SQL host program> prior to any <embedded SQL statement> that references the <host identifier>. The <host variable definition> shall be such that a host language reference to the <host identifier> is valid at every <embedded SQL statement> that contains the <host identifier>.

18) A <host variable definition> defines the host language data type of the <host identifier>. For every such host language data type an equivalent SQL <data type> is specified in Subclause 16.3, "<embedded SQL Ada program>", Subclause 16.4, "<embedded SQL C program>", Subclause 16.5, "<embedded SQL COBOL program>", Subclause 16.6, "<embedded SQL Fortran program>", Subclause 16.7, "<embedded SQL MUMPS program>", Subclause 16.8, "<embedded SQL Pascal program>", and Subclause 16.9, "<embedded SQL PL/I program>".

19) An <embedded SQL host program> shall contain a <host variable definition> that specifies SQLSTATE.

20) If one or more <host variable definition>s that specify SQLSTATE appear in an <embedded SQL host program>, then the <host variable definition>s shall be such that a host language reference to SQLSTATE is valid at every <embedded SQL statement>, including <embedded SQL statement>s that appear in any subprograms contained in that <embedded SQL host program>. The first such <host variable definition> of SQLSTATE shall appear in the text of the <embedded SQL host program> prior to any <embedded SQL statement>.

21) Given an <embedded SQL host program> *H*, there is an implied standard-conforming SQL-client module *M* and an implied standard-conforming host program *P* derived from *H*. The derivation of the implied program *P* and the implied <SQL-client module definition> *M* of an <embedded SQL host program> *H* effectively precedes the processing of any host language program text manipulation commands such as inclusion or copying of text.

    Given an <embedded SQL host program> *H* with an implied <SQL-client module definition> *M* and an implied program *P* defined as above:

    a) The implied <SQL-client module definition> *M* of *H* shall be a standard-conforming <SQL-client module definition>.

    b) If *H* is an <embedded SQL Ada program>, an <embedded SQL C program>, an <embedded SQL COBOL program>, an <embedded SQL Fortran program>, an <embedded SQL MUMPS program>, an <embedded SQL Pascal program>, or an <embedded SQL PL/I program>, then the implied program *P* shall be a standard-conforming Ada program, a standard-conforming C program, a standard-conforming COBOL program, a standard-conforming Fortran program, a standard-conforming MUMPS program, a standard-conforming Pascal program, or standard-conforming PL/I program, respectively.

22) *M* is derived from *H* as follows:

a) *M* contains a <module name clause> whose <module name> is either implementation-dependent or is omitted.

b) *M* contains a <module character set specification> that is identical to the explicit or implicit <embedded character set declaration> with the keyword "SQL" removed.

c) *M* contains a <language clause> that specifies either ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, where *H* is respectively an <embedded SQL Ada program>, an <embedded SQL C program>, an <embedded SQL COBOL program>, an <embedded SQL Fortran program>, an <embedded SQL MUMPS program>, an <embedded SQL Pascal program>, or an <embedded SQL PL/I program>.

d) Case:

i) If *H* contains an <embedded authorization declaration> *EAD*, then let *EAC* be the <embedded authorization clause>contained in *EAD*; *M* contains a <module authorization clause> that specifies *EAC*.

ii) Otherwise, let *SN* be an implementation-defined <schema name>; *M* contains a <module authorization clause> that specifies "SCHEMA *SN*".

e) Case:

i) If *H* contains an <embedded path specification> *EPS*, then *M* contains the <module path specification> *EPS*.

ii) Otherwise, *M* contains am implementation-defined <module path specification>.

f) *M* contains a <module transform group specification> that is identical to the explicit or implicit <embedded transform group specification>.

g) For every <declare cursor> *EC* contained in *H*, *M* contains one <declare cursor> *PC* and one <externally-invoked procedure> *PS* that contains an <open statement> that references *PC*.

i) The <procedure name> of *PS* is implementation-dependent. *PS* contains a <host parameter declaration> *PD* for each distinct <embedded variable name> *EVN* contained in *PC* with an implementation-dependent <host parameter name> *PN* and the <host parameter data type> *PT*, determined as follows:

Case:

1) If *EVN* identifies a binary large object locator variable, then *PT* is BLOB AS LOCATOR.

2) If *EVN* identifies a character large object locator variable, then *PT* is CLOB AS LOCATOR.

3) If *EVN* identifies an array locator variable, then *PT* is *AAT* AS LOCATOR, where *AAT* is the associated array type of *V*.

4) If *EVN* identifies a user-defined type locator variable, then *PT* is *UDT* AS LOCATOR, where *UDT* is the associated user-defined type of *V*.

5) Otherwise, *PT* is the SQL data type that corresponds to the host language data type of *EVN* as specified in Subclause 13.6, "Data type correspondences", in ISO/IEC 9075-2.

ii) *PS* contains a <host parameter declaration> that specifies SQLSTATE . The order of <host parameter declaration>s in *PS* is implementation-dependent. *PC* is a copy of *EC* in which each *EVN* has been replaced as follows:

Case:

1) If *EVN* does not identify user-defined type locator variable, but *EVN* identifies a host variable that has an associated user-defined type *UT*, then:

A) Let *GN* be the <group name> corresponding to the <user-defined type name> of *UT* contained in <group specification> contained in <embedded group specification>.

B) Apply the Syntax Rules of Subclause 10.17, "Determination of a to-sql function", in ISO/IEC 9075-2, with *DT* and *GN* as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function *TSF*.

C) Let the declared type of the single SQL parameter of *TSF* be *TPT*. *PT* and *TPT* shall be mutually assignable.

D) *EVN* is replaced by:

```
TSFN(CAST(PN AS TPT))
```

2) Otherwise, *EVN* is replaced by:

```
PN
```

h) For every <dynamic declare cursor> *EC* in *H*, *M* contains one <dynamic declare cursor> *PC* that is a copy of *EC*.

i) *M* contains one <temporary table declaration> for each <temporary table declaration> contained in *H*. Each <temporary table declaration> of *M* is a copy of the corresponding <temporary table declaration> of *H*.

j) *M* contains one <embedded exception declaration> for each <embedded exception declaration> contained in *H*. Each <embedded exception declaration> of *M* is a copy of the corresponding <embedded exception declaration> of *H*.

k) *M* contains an <externally-invoked procedure> for each <SQL procedure statement> contained in *H*. The <externally-invoked procedure> *PS* of *M* corresponding with an <SQL procedure statement> *ES* of *H* is defined as follows.

Case:

i) If *ES* is not an <open statement>, then:

1) The <procedure name> of *PS* is implementation-dependent.

2) Let n be the number of distinct <embedded variable name>s contained in ES. Let HVN_i, 1 (one) $\leq i \leq n$, be the i-th such <embedded variable name> and let HV_i be the host variable identified by HVN_i.

3) For each HVN_i, 1 (one) $\leq i \leq n$, PS contains a <host parameter declaration> PD_i defining a host parameter P_i such that:

   A) The <host parameter name> PN_i of PD_i is implementation-dependent.

   B) The <host parameter data type> PT_i of PD_i is determined as follows:

      I) If HV_i is a binary large object locator variable, then PT_i is BLOB AS LOCATOR.

      II) If HV_i is a character large object locator variable, then PT_i is CLOB AS LOCATOR.

      III) If HV_i is an array locator variable, then PT_i is AAT AS LOCATOR, where AAT is the associated array type of HV_i.

      IV) If HV_i is user-defined type locator variable, then PT_i is UDT AS LOCATOR, where UDT is the associated user-defined type of HV_i.

      V) Otherwise, PT_i is the SQL data type that corresponds to the host language data type of HV_i as specified in Subclause 13.6, "Data type correspondences", in ISO/IEC 9075-2.

4) PS contains a <host parameter declaration> that specifies SQLSTATE.

5) The order of the <host parameter declaration>s PD_i, 1 (one) $\leq i \leq n$, is implementation-dependent.

6) For each HVN_i, 1 (one) $\leq i \leq n$, that identifies some HV_i that has an associated user-defined type, but is not a user-defined type locator variable, apply the Syntax Rules of Subclause 9.6, "Host parameter mode determination", in ISO/IEC 9075-2, with the PD_i corresponding to HVN_i and ES as <host parameter declaration> and <SQL procedure statement>, respectively, to determine whether the corresponding P_i is an input host parameter, an output host parameter, or both an input host parameter and an output host parameter.

   A) Among P_i, 1 (one) $\leq i \leq n$, let a be the number of input host parameters, b be the number of output host parameters, and let c be the number of host parameters that are both input host parameters and output host parameters.

   B) Among P_i, 1 (one) $\leq i \leq n$, let PI_j, 1 (one) $\leq j \leq a$, be the input host parameters, let PO_k, 1 (one) $\leq k \leq b$, be the output host parameters, and let PIO_l, 1 (one) $\leq l \leq c$, be the host parameters that are both input host parameters and output host parameters.

   C) Let PNI_j, 1 (one) $\leq j \leq a$, be the <host parameter name> of PI_j. Let PNO_k, 1 (one) $\leq k \leq b$, be the <host parameter name> of PO_k. Let $PNIO_l$, 1 (one) $\leq l \leq c$, be the <host parameter name> of PIO_l.

   D) Let HVI_j, 1 (one) $\leq j \leq a$, be the host variable corresponding to PI_j. Let HVO_k, 1 (one) $\leq k \leq b$, be the host variable corresponding to PO_k. Let $HVIO_l$, 1 (one) $\leq l \leq c$, be the host variable corresponding to PIO_l.

E) Let TSI_j, 1 (one) $\leq j \leq a$, be the associated SQL data types of HVI_j. Let TSO_k, 1 (one) $\leq k \leq b$, be the associated SQL data types of HVO_k. Let $TSIO_l$, 1 (one) $\leq l \leq c$, be the associated SQL data types of $HVIO_l$.

F) Let TUI_j, 1 (one) $\leq j \leq a$, be the associated user-defined types of HVI_j. Let TUO_k, 1 (one) $\leq k \leq b$, be the associated user-defined types of HVO_k. Let $TUIO_l$, 1 (one) $\leq l \leq c$, be the associated user-defined types of $HVIO_l$.

G) Let GNI_j, 1 (one) $\leq j \leq a$, be the <group name>s corresponding to the <user-defined type name> of TUI_j contained in the <group specification> contained in <embedded group specification>. Let GNO_k, 1 (one) $\leq k \leq b$, be the <group name>s corresponding to the <user-defined type name> of TUO_k contained in the <group specification> contained in <embedded group specification>. Let $GNIO_l$, 1 (one) $\leq l \leq c$, be the <group name>s corresponding to the <user-defined type name> of $TUIO_l$ contained in the <group specification> contained in <embedded group specification>.

H) For every j, 1 (one) $\leq j \leq a$, apply the Syntax Rules of Subclause 10.17, "Determination of a to-sql function", in ISO/IEC 9075-2, with TUI_j and GNI_j as $TYPE$ and $GROUP$, respectively. There shall be an applicable to-sql function $TSFI_j$ identified by <routine name> $TSIN_j$. Let TTI_j be the data type of the single SQL parameter of $TSFI_j$. TTI_j and TSI_j shall be mutually assignable.

I) For every l, 1 (one) $\leq l \leq c$, apply the Syntax Rules of Subclause 10.17, "Determination of a to-sql function", in ISO/IEC 9075-2, with $TUIO_l$ and $GNIO_l$ as $TYPE$ and Subclause 7.9, "<group by clause>", respectively. There shall be an applicable to-sql function $TSFIO_l$ identified by <routine name> $TSION_l$. Let $TTIO_l$ be the data type of the single SQL parameter of $TSFIO_l$. $TTIO_l$ and $TSIO_l$ shall be mutually assignable.

J) For every k, 1 (one) $\leq k \leq b$, apply the Syntax Rules of Subclause 10.15, "Determination of a from-sql function", in ISO/IEC 9075-2, with TUO_k and GNO_k as $TYPE$ and $GROUP$, respectively. There shall be an applicable from-sql function $FSFO_k$ identified by <routine name> $FSON_k$. Let TRO_k be the result data type of $FSFO_k$. TRO_k and TSO_k shall be mutually assignable.

K) For every l, 1 (one) $\leq l \leq c$, apply the Syntax Rules of Subclause 10.15, "Determination of a from-sql function", in ISO/IEC 9075-2, with $TUIO_l$ and $GNIO_l$ as $TYPE$ and $GROUP$, respectively. There shall be an applicable from-sql function $FSFIO_l$ identified by <routine name> $FSION_l$. Let $TRIO_l$ be the result data type of $FSFIO_l$. $TRIO_l$ and $TSIO_l$ shall be mutually assignable.

L) Let SVI_j, 1 (one) $\leq j \leq a$, SVO_k, 1 (one) $\leq k \leq b$, and $SVIO_l$, 1 (one) $\leq l \leq c$, be implementation-dependent <SQL variable name>s, each of which is not equivalent to any other <SQL variable name> contained in ES, to any <SQL parameter name> contained in ES, or to any <column name> contained in ES.

7) Let NES be an <SQL procedure statement> that is a copy of ES in which every HVN_i, 1 (one) $\leq i \leq n$, is replaced as follows.

Case:

A) If HV_i has an associated user-defined type but is not a user-defined type locator variable, then

Case:

I) If P_i is an input host parameter, then let PI_j, 1 (one) $\leq j \leq a$, be the input host parameter that corresponds to P_i; HVN_i is replaced by SVI_j.

II) If P_i is an output host parameter, then let PO_k, 1 (one) $\leq k \leq b$, be the output host parameter that corresponds to P_i; HVN_i is replaced by SVO_k.

III) Otherwise, let PIO_l, 1 (one) $\leq l \leq c$, be the intput host parameter and the output host parameter that corresponds to P_i; HVN_i is replaced by $SVIO_l$.

B) Otherwise, HVN_i is replaced by PN_i.

8) The <SQL procedure statement> of *PS* is:

```
BEGIN ATOMIC
    DECLARE SVI_1 TUI_1;
    .
    .
    .
    DECLARE SVI_a TUI_a;
    DECLARE SVO_1 TUO_1;
    .
    .
    .
    DECLARE SVO_a TUO_a;
    DECLARE SVIO_1 TUIO_1;
    .
    .
    .
    DECLARE SVIO_a TUIO_a;
    SET SVI_1 = TSIN_1 (CAST (PNI_1 AS TTI_1));
    .
    .
    .
    SET SVI_a = TSIN_a (CAST (PNI_a AS TTI_a));
    SET SVIO_1 = TSION_1 (CAST (PNIO_1 AS TTIO_1));
    .
    .
    .
    SET SVIO_c = TSION_c (CAST (PNIO_c AS TTIO_c));
    NES;
    SET PNO_1 = CAST ( FSON_1 (SVO_1) AS TSO_1);
    .
    .
    .
    SET PNO_b = CAST ( FSON_b (SVO_b) AS TSO_b);
    SET PNIO_1 = CAST ( FSION_1 (SVIO_1) AS TSIO_1);
    .
    .
    .
    SET PNIO_c = CAST ( FSION_c (SVIO_c) AS TSIO_c);
END;
```

9) Whether one <externally-invoked procedure> of *M* can correspond to more than one <SQL procedure statement> of *H* is implementation-dependent.

ii) If *ES* is an <open statement>, then:

1) Let *EC* be the <declare cursor> in *H* referenced by *ES*.

2) *PS* is the <externally-invoked procedure> in *M* that contains an <open statement> that references the <declare cursor> in M corresponding to *EC*.

23) *P* is derived from *H* as follows:

a) Each <embedded SQL begin declare>, <embedded SQL end declare>, and <embedded character set declaration> has been deleted. If the embedded host language is MUMPS, then each <embedded SQL MUMPS declare> has been deleted.

b) Each <host variable definition> in an <embedded SQL declare section> has been replaced by a valid data definition in the target host language according to the Syntax Rules specified in an <embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL COBOL program>, <embedded SQL Fortran program>, <embedded SQL Pascal program>, or an <embedded SQL PL/I program> clause.

c) Each <embedded SQL statement> that contains a <declare cursor>, a <dynamic declare cursor>, an <SQL-invoked routine>, or a <temporary table declaration> has been deleted, and every <embedded SQL statement> that contains an <embedded exception declaration> has been replaced with statements of the host language that will have the effect specified by the General Rules of Subclause 16.2, "<embedded exception declaration>".

d) Each <embedded SQL statement> that contains an <SQL procedure statement> has been replaced by host language statements that perform the following actions:

i) A host language procedure or subroutine call of the <externally-invoked procedure> of the implied <SQL-client module definition> *M* of *H* that corresponds with the <SQL procedure statement>.

If the <SQL procedure statement> is not an <open statement>, then the arguments of the call include each distinct <host identifier> contained in the <SQL procedure statement> together with the SQLSTATE <host identifier>. If the <SQL procedure statement> is an <open statement>, then the arguments of the call include each distinct <host identifier> contained in the corresponding <declare cursor> of *H* together with the SQLSTATE <host identifier>.

The order of the arguments in the call corresponds with the order of the corresponding <host parameter declaration>s in the corresponding <externally-invoked procedure>.

NOTE 25 – In an <embedded SQL Fortran program>, the "SQLSTATE" variable may be abbreviated to "SQLSTA". See the Syntax Rules of Subclause 16.6, "<embedded SQL Fortran program>".

ii) Exception actions, as specified in Subclause 16.2, "<embedded exception declaration>".

e) Each <statement or declaration> that contains an <embedded authorization declaration> is deleted.

Access Rules

1) For every host variable whose <embedded variable name> is contained in <statement or declaration> and has an associated user-defined type, the current privileges shall include EXECUTE privilege on all from-sql functions (if any) and all to-sql functions (if any) referenced in the corresponding SQL-client module.

General Rules

1) The interpretation of an <embedded SQL host program> *H* is defined to be equivalent to the interpretation of the implied program *P* of *H* and the implied <SQL-client module definition> *M* of *H*.

2) If the cursor mode of the current SQL-transaction is *cascade off*, and the <embedded SQL statement> is other than a <close statement>, a <delete statement: positioned>, a <fetch statement>, an <open statement>, an <update statement: positioned>, a <dynamic close statement>, a <dynamic delete statement: positioned>, a <dynamic fetch statement>, a <dynamic open statement>, or a <dynamic update statement: positioned>, then an exception condition is raised: *invalid SQL statement.*

Conformance Rules

1) Without Feature B051, “Enhanced execution rights”, conforming SQL language shall not contain any <embedded authorization declaration>.

2) Without Feature F451, “Character set definition”, or Feature F461, “Named character sets”, an <embedded SQL declare section> shall not contain an <embedded character set declaration>.

3) Without Feature F361, “Subprogram support”, no two <host variable definition>s shall specify the same variable name.

4) Without Feature S071, “SQL paths in function and type name resolution”, <embedded path specification> shall not be specified.

5) Without Feature S241, “Transform functions”, <embedded transform group specification> shall not be specified.

16.2 <embedded exception declaration>

Function

Specify the action to be taken when an SQL-statement causes a specific class of condition to be raised.

Format

```
<embedded exception declaration> ::=
      WHENEVER <condition> <condition action>

<condition> ::=
      <SQL condition>

<SQL condition> ::=
        <major category>
      | SQLSTATE ( <SQLSTATE class value> [ , <SQLSTATE subclass value> ] )
      | CONSTRAINT <constraint name>

<major category> ::=
        SQLEXCEPTION
      | SQLWARNING
      | NOT FOUND

<SQLSTATE class value> ::=
      <SQLSTATE char><SQLSTATE char> !! See the Syntax Rules.

<SQLSTATE subclass value> ::=
      <SQLSTATE char><SQLSTATE char><SQLSTATE char> !! See the Syntax Rules.

<SQLSTATE char> ::= <simple Latin upper case letter> | <digit>

<condition action> ::=
      CONTINUE | <go to>

<go to> ::=
      { GOTO | GO TO } <goto target>

<goto target> ::=
        <host label identifier>
      | <unsigned integer>
      | <host PL/I label variable>

<host label identifier> ::= !! See the Syntax Rules.

<host PL/I label variable> ::= !! See the Syntax Rules.
```

Syntax Rules

1) SQLWARNING, NOT FOUND, and SQLEXCEPTION correspond to SQLSTATE class values corresponding to categories W, N, and X in Table 10, “SQLSTATE class and subclass values”, respectively.

2) An <embedded exception declaration> contained in an <embedded SQL host program> applies to an <SQL procedure statement> contained in that <embedded SQL host program> if and only if the <SQL procedure statement> appears after the <embedded exception declaration> that has condition *C* in the text sequence of the <embedded SQL host program> and no other <embedded exception declaration> *E* that satisfies one of the following conditions appears between the <embedded exception declaration> and the <SQL procedure statement> in the text sequence of the <embedded SQL host program>.

   Let *D* be the <condition> contained in *E*.

   Case:

   a) *D* is the same as *C*.

   b) *D* is a <major category> and belongs to the same class to which *C* belongs.

   c) *D* contains an <SQLSTATE class value>, but does not contain an <SQLSTATE subclass value>, and *E* contains the same <SQLSTATE class value> that *C* contains.

   d) *D* contains the <SQLSTATE class value> that corresponds to *integrity constraint violation* and *C* contains CONSTRAINT.

3) In the values of <SQLSTATE class value> and <SQLSTATE subclass value>, there shall be no <separator> between the <SQLSTATE char>s.

4) The values of <SQLSTATE class value> and <SQLSTATE subclass value> shall correspond to class values and subclass values, respectively, specified in Table 10, “SQLSTATE class and subclass values”.

5) If an <embedded exception declaration> specifies a <go to>, then the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> shall be such that a host language GO TO statement specifying that <host label identifier>, <host PL/I label variable>, or <unsigned integer> is valid at every <SQL procedure statement> to which the <embedded exception declaration> applies.

   NOTE 26 –

   If an <embedded exception declaration> is contained in an <embedded SQL Ada program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a label_name in the containing <embedded SQL Ada program>.
   If an <embedded exception declaration> is contained in an <embedded SQL C program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a label in the containing <embedded SQL C program>.
   If an <embedded exception declaration> is contained in an <embedded SQL COBOL program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a section-name or an unqualified paragraph-name in the containing <embedded SQL COBOL program>.
   If an <embedded exception declaration> is contained in an <embedded SQL Fortran program>, then the <goto target> of a <go to> should be an <unsigned integer> that is the statement label of an executable statement that appears in the same program unit as the <go to>.
   If an <embedded exception declaration> is contained in an <embedded SQL MUMPS program>, then the <goto target> of a <go to> should be a gotoargument that is the statement label of an executable statement that appears in the same <embedded SQL MUMPS program>.
   If an <embedded exception declaration> is contained in an <embedded SQL Pascal program>, then the <goto target> of a <go to> should be an <unsigned integer> that is a label.
   If an <embedded exception declaration> is contained in an <embedded SQL PL/I program>, then the <goto target> of a <go to> should specify either a <host label identifier> or a <host PL/I label variable>.

Case:

— If <host label identifier> is specified, then the <host label identifier> should be a label constant in the containing <embedded SQL PL/I program>.

— If <host PL/I label variable> is specified, then the <host PL/I label variable> should be a PL/I label variable declared in the containing <embedded SQL PL/I program>.

Access Rules

None.

General Rules

1) Immediately after the execution of an <SQL procedure statement> *STMT* in an <embedded SQL host program> that returns an SQLSTATE value other than *successful completion*:

   a) Let *E* be the set of <embedded exception declaration>s that are contained in the <embedded SQL host program> containing *STMT*, that applies to *STMT*, and that specifies an <condition action> that is <go to>.

   b) Let *CV* and *SCV* be respectively the values of the class and subclass of the SQLSTATE value that indicates the result of the <SQL procedure statement>.

   c) If the execution of the <SQL procedure statement> caused the violation of one or more constraints or assertions, then:

      i) Let *ECN* be the set of <embedded exception declaration>s in *E* that specify CONSTRAINT and the <constraint name> of a constraint that was violated by execution of *STMT*.

      ii) If *ECN* contains more than one <embedded exception declaration>, then an implementation-dependent <embedded exception declaration> is chosen from *ECN*; otherwise, the single <embedded exception declaration> in *ECN* is chosen.

      iii) A GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> chosen from *ECN*.

   d) Otherwise:

      i) Let *ECS* be the set of <embedded exception declaration>s in *E* that specify SQLSTATE, an <SQLSTATE class value>, and an <SQLSTATE subclass value>.

      ii) If *ECS* contains an <embedded exception declaration> *EY* that specifies an <SQLSTATE class value> identical to *CV* and an <SQLSTATE subclass value> identical to *SCV*, then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

      iii) Otherwise:

         1) Let *EC* be the set of <embedded exception declaration>s in *E* that specify SQLSTATE and an <SQLSTATE class value> without an <SQLSTATE subclass value>.

2) If *EC* contains an <embedded exception declaration> *EY* that specifies an <SQLSTATE class value> identical to *CV*, then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

3) Otherwise:

   A) Let *EX* be the set of <embedded exception declaration>s in *E* that specify SQLEXCEPTION.

   B) If *EX* contains an <embedded exception declaration> *EY* and *CV* belongs to Category X in Table 10, "SQLSTATE class and subclass values", then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

   C) Otherwise:

      I) Let *EW* be the set of <embedded exception declaration>s in *E* that specify SQLWARNING.

      II) If *EW* contains an <embedded exception declaration> *EY* and *CV* belongs to Category W in Table 10, "SQLSTATE class and subclass values", then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

      III) Otherwise, let *ENF* be the set of <embedded exception declaration>s in *E* that specify NOT FOUND. If *ENF* contains an <embedded exception declaration> *EY* and *CV* belongs to Category N in Table 10, "SQLSTATE class and subclass values", then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> specified in the <embedded exception declaration> *EY*.

Conformance Rules

1) Without Feature B041, "Extensions to embedded SQL exception declarations", an <SQL condition> shall not specify SQLSTATE or CONSTRAINT.

2) Without Feature B041, "Extensions to embedded SQL exception declarations", a <major category> shall not specify SQLEXCEPTION or SQLWARNING.

16.3 <embedded SQL Ada program>

Function

Specify an <embedded SQL Ada program>.

Format

```
<embedded SQL Ada program> ::= !! See the Syntax Rules.

<Ada variable definition> ::=
      <Ada host identifier> [ { <comma> <Ada host identifier> }... ] <colon>
      <Ada type specification> [ <Ada initial value> ]

<Ada initial value> ::=
      <Ada assignment operator> <character representation>...

<Ada assignment operator> ::= <colon><equals operator>

<Ada host identifier> ::= !! See the Syntax Rules.

<Ada type specification> ::=
        <Ada qualified type specification>
      | <Ada unqualified type specification>
      | <Ada derived type specification>

<Ada qualified type specification> ::=
        Interfaces.SQL <period> CHAR [ CHARACTER SET [ IS ] <character set specification> ]

            <left paren> 1 <double period> <length> <right paren>
      | Interfaces.SQL <period> BIT <left paren> 1 <double period> <length> <right paren>

      | Interfaces.SQL <period> SMALLINT
      | Interfaces.SQL <period> INT
      | Interfaces.SQL <period> REAL
      | Interfaces.SQL <period> DOUBLE_PRECISION
      | Interfaces.SQL <period> BOOLEAN
      | Interfaces.SQL <period> SQLSTATE_TYPE
      | Interfaces.SQL <period> INDICATOR_TYPE

<Ada unqualified type specification> ::=
        CHAR <left paren> 1 <double period> <length> <right paren>
      | BIT <left paren> 1 <double period> <length> <right paren>
      | SMALLINT
      | INT
      | REAL
      | DOUBLE_PRECISION
      | BOOLEAN
      | SQLSTATE_TYPE
      | INDICATOR_TYPE

<Ada derived type specification> ::=
        <Ada CLOB variable>
      | <Ada BLOB variable>
      | <Ada user-defined type variable>
      | <Ada CLOB locator variable>
      | <Ada BLOB locator variable>
      | <Ada user-defined type locator variable>
      | <Ada array locator variable>
      | <Ada REF variable>
```

```
<Ada CLOB variable> ::=
      SQL TYPE IS CLOB <left paren> <large object length> <right paren>
        [ CHARACTER SET [ IS ] <character set specification> ]

<Ada BLOB variable> ::=
      SQL TYPE IS BLOB <left paren> <large object length> <right paren>

<Ada user-defined type variable> ::=
      SQL TYPE IS <user-defined type> AS <predefined type>

<Ada CLOB locator variable> ::=
      SQL TYPE IS CLOB AS LOCATOR

<Ada BLOB locator variable> ::=
      SQL TYPE IS BLOB AS LOCATOR

<Ada user-defined type locator variable> ::=
      SQL TYPE IS <user-defined type name> AS LOCATOR

<Ada array locator variable> ::=
      SQL TYPE IS <collection type> AS LOCATOR

<Ada REF variable> ::=
      SQL TYPE IS <reference type>
```

Syntax Rules

1) An <embedded SQL Ada program> is a compilation unit that consists of Ada text and SQL text. The Ada text shall conform to the Ada standard ISO/IEC 8652. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever an Ada statement may be specified. An <embedded SQL statement> may be prefixed by an Ada label.

3) An <Ada host identifier> is any valid Ada identifier. An <Ada host identifier> shall be contained in an <embedded SQL Ada program>.

4) An <Ada variable definition> defines one or more host variables.

5) An <Ada variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Ada program> (see the Syntax Rules of Subclause 16.1, "<embedded SQL host program>"):

   a) Any optional CHARACTER SET specification shall be removed from an <Ada qualified type specification> and <Ada derived type specification>.

   b) The <length> specified in a CHAR declaration of any <Ada qualified type specification> or <Ada derived type specification> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Ada host identifier> specified in the containing <Ada variable definition>.

   c) The syntax

      SQL TYPE IS CLOB (*L*)

and the syntax

```
SQL TYPE IS BLOB ( L )
```

for a given <Ada host identifier> *HVN* shall be replaced by

```
TYPE HVN IS RECORD
  HVN_RESERVED : Interfaces.SQL.INT;
  HVN_LENGTH : Interfaces.SQL.INT;
  HVN_DATA : Interfaces.SQL.CHAR(1..L);
END RECORD;
```

in any <Ada CLOB variable> or <Ada BLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

d) The syntax

```
SQL TYPE IS UDTN AS PDT
```

shall be replaced by

```
ADT
```

in any <Ada user-defined type variable>, where *ADT* is the data type listed in the “Ada data type” column corresponding to the row for SQL data type *PDT* in Table 18, "Data type correspondences for Ada", in ISO/IEC 9075-2. *ADT* shall not be “none”. The data type identified by *UDTN* is called the *associated user-defined type* of the host variable and the data type identified by *PDT* is called the *associated SQL data type* of the host variable.

e) The syntax

```
SQL TYPE IS BLOB AS LOCATOR
```

shall be replaced by

```
Interfaces.SQL.INT
```

in any <Ada BLOB locator variable>. The host variable defined by <Ada BLOB locator variable> is called a *binary large object locator variable.*

f) The syntax

```
SQL TYPE IS CLOB AS LOCATOR
```

shall be replaced by

```
Interfaces.SQL.INT
```

in any <Ada CLOB locator variable>. The host variable defined by <Ada CLOB locator variable> is called a *character large object locator variable.*

g) The syntax

```
SQL TYPE IS <user-defined type> AS LOCATOR
```

shall be replaced by

```
Interfaces.SQL.INT
```

in any <Ada user-defined type locator variable>. The host variable defined by <Ada user-defined type locator variable> is called a *user-defined type locator variable*. The data type identified by <user-defined type> is called the *associated user-defined type* of the host variable.

h) The syntax

```
SQL TYPE IS <collection type> AS LOCATOR
```

shall be replaced by

```
Interfaces.SQL.INT
```

in any <Ada array locator variable>. The host variable defined by <Ada array locator variable> is called an *array locator variable*. The data type identified by <collection type> is called the *associated array type* of the host variable.

i) The syntax

```
SQL TYPE IS <reference type>
```

for a given <Ada host identifier> *RTV* shall be replaced by

```
RTV : Interfaces.SQL.CHAR(1..<length>)
```

in any <Ada reference type>, where <length> is the implementation-defined length in octets of a reference type.

The modified <Ada variable definition> shall be a valid Ada object-declaration in the program derived from the <embedded SQL Ada program>.

6) The reference type identified by <reference type> contained in an <Ada REF variable> is called the *referenced type* of the reference.

7) An <Ada variable definition> shall be specified within the scope of Ada **with** and **use** clauses that specify the following:

with Interfaces.SQL;

use Interfaces.SQL;

use Interfaces.SQL.CHARACTER_SET;

8) The <character representation> sequence in an <Ada initial value> specifies an initial value to be assigned to the Ada variable. It shall be a valid Ada specification of an initial value.

9) CHAR describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

10) BIT describes a bit string variable. The equivalent SQL data type is BIT with the same length.

11) INT and SMALLINT describe exact numeric variables. The equivalent SQL data types are INTEGER and SMALLINT, respectively.

12) REAL and DOUBLE_PRECISION describe approximate numeric variables. The equivalent SQL data types are REAL and DOUBLE PRECISION, respectively.

13) BOOLEAN describes a boolean variable. The equivalent SQL data type is BOOLEAN.

14) SQLSTATE_TYPE describes a character string variable whose length is the length of the SQLSTATE parameter, five characters.

15) INDICATOR_TYPE describes an exact numeric variable whose specific data type is any <exact numeric type> with a scale of 0 (zero).

Access Rules

None.

General Rules

1) See Subclause 16.1, "<embedded SQL host program>".

Conformance Rules

1) Without Feature B011, "Embedded Ada", <embedded SQL Ada program> shall not be specified.

2) Without Feature F511, "BIT data type", an <Ada variable definition> shall not specify a bit string variable.

3) Without Feature F451, "Character set definition", or Feature F461, "Named character sets", an <Ada qualified type specification> shall not contain a <character set specification>.

4) Without Feature S041, "Basic reference types", an <Ada derived type specification> shall not be an <Ada REF variable>.

5) Without Feature S241, "Transform functions", an <Ada derived type specification> shall not be an <Ada user-defined type variable>.

6) Without Feature S232, "Array locators", an <Ada derived type specification> shall not be an <Ada array locator variable>.

7) Without Feature S231, "Structured type locators", the <user-defined type name> simply contained in an <Ada user-defined type locator variable> shall not identify a structured type.

8) Without Feature T041, "Basic LOB data type support", an <Ada BLOB variable>, <Ada CLOB variable>, <Ada BLOB locator variable>, and <Ada CLOB locator variable> shall not be specified.

16.4 <embedded SQL C program>

Function

Specify an <embedded SQL C program>.

Format

```
<embedded SQL C program> ::= !! See the Syntax Rules.

<C variable definition> ::=
          [ <C storage class> ]
          [ <C class modifier> ]
          <C variable specification>
        <semicolon>

<C variable specification> ::=
          <C numeric variable>
        | <C character variable>
        | <C derived variable>

<C storage class> ::=
          auto
        | extern
        | static

<C class modifier> ::=
          const
        | volatile

<C numeric variable> ::=
        { long | short | float | double }
          <C host identifier> [ <C initial value> ]
                [ { <comma> <C host identifier> [ <C initial value> ] }... ]

<C character variable> ::=
        <C character type> [ CHARACTER SET [ IS ] <character set specification> ]
          <C host identifier> <C array specification> [ <C initial value> ]
            [ { <comma> <C host identifier> <C array specification>
                      [ <C initial value> ] }... ]

<C character type> ::=
          char
        | unsigned char
        | unsigned short

<C array specification> ::=
        <left bracket> <length> <right bracket>

<C host identifier> ::= !! See the Syntax Rules.

<C derived variable> ::=
          <C VARCHAR variable>
        | <C NCHAR variable>
        | <C NCHAR VARYING variable>
        | <C CLOB variable>
        | <C NCLOB variable>
        | <C BLOB variable>
        | <C bit variable>
```

```
      | <C user-defined type variable>
      | <C CLOB locator variable>
      | <C BLOB locator variable>
      | <C array locator variable>
      | <C user-defined type locator variable>
      | <C REF variable>

<C VARCHAR variable> ::=
      VARCHAR [ CHARACTER SET [ IS ] <character set specification> ]
        <C host identifier> <C array specification> [ <C initial value> ]
          [ { <comma> <C host identifier> <C array specification> [ <C initial value> ] }... ]

<C NCHAR variable> ::=
      NCHAR [ CHARACTER SET [ IS ] <character set specification> ]
        <C host identifier> <C array specification> [ <C initial value> ]
          [ { <comma> <C host identifier> <C array specification> [ <C initial value> ] } ... ]

<C NCHAR VARYING variable> ::=
      NCHAR VARYING [ CHARACTER SET [ IS ] <character set specification> ]
        <C host identifier> <C array specification> [ <C initial value> ]
          [ { <comma> <C host identifier> <C array specification> [ <C initial value> ] } ... ]

<C CLOB variable> ::=
      SQL TYPE IS CLOB <left paren> <large object length> <right paren>
        [ CHARACTER SET [ IS ] <character set specification> ]
        <C host identifier> [ <C initial value> ]
          [ { <comma> <C host identifier> [ <C initial value> ] }... ]

<C NCLOB variable> ::=
      SQL TYPE IS NCLOB <left paren> <large object length> <right paren>
        [ CHARACTER SET [ IS ] <character set specification> ]
        <C host identifier> [ <C initial value> ]
          [ { <comma> <C host identifier> [ <C initial value> ] }... ]

<C bit variable> ::=
      BIT <C host identifier> <C array specification> [ <C initial value> ]
        [ { <comma> <C host identifier> <C array specification>
                  [ <C initial value> ] }... ]

<C user-defined type variable> ::=
      SQL TYPE IS <user-defined type> AS <predefined type>
        <C host identifier> [ <C initial value> ]
          [ { <comma> <C host identifier> [ <C initial value> ] } ... ]

<C BLOB variable> ::=
      SQL TYPE IS BLOB <left paren> <large object length> <right paren>
        <C host identifier> [ <C initial value> ]
          [ { <comma> <C host identifier> [ <C initial value> ] } ... ]

<C CLOB locator variable> ::=
      SQL TYPE IS CLOB AS LOCATOR
        <C host identifier> [ <C initial value> ]
          [ { <comma> <C host identifier> [ <C initial value> ] } ... ]

<C BLOB locator variable> ::=
        SQL TYPE IS BLOB AS LOCATOR
        <C host identifier> [ <C initial value> ]
          [ { <comma> <C host identifier> [ <C initial value> ] } ... ]
```

```
<C array locator variable> ::=
         SQL TYPE IS <collection type> AS LOCATOR
         <C host identifier> [ <C initial value> ]
           [ { <comma> <C host identifier> [ <C initial value> ] } ... ]

<C user-defined type locator variable> ::=
       SQL TYPE IS
           <user-defined type> AS LOCATOR
       <C host identifier> [ <C initial value> ]
       [ { <comma> <C host identifier> [ <C initial value> ] }... ]

<C REF variable> ::=
         SQL TYPE IS <reference type>

<C initial value> ::=
       <equals operator> <character representation>...
```

Syntax Rules

1) An <embedded SQL C program> is a compilation unit that consists of C text and SQL text. The C text shall conform to the C standard ISO/IEC 9899. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a C statement may be specified within a function block. If the C statement could include a label prefix, then the <embedded SQL statement> may be immediately preceded by a label prefix.

3) A <C host identifier> is any valid C variable identifier. A <C host identifier> shall be contained in an <embedded SQL C program>.

4) A <C variable definition> defines one or more host variables.

5) The <collection type> contained in a <C array locator variable> shall specify an array data type. The array data type specified by the <collection type> contained in a <C array locator variable> is called the *associated array data type* of the array locator.

6) A <C variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL C program> (see the Syntax Rules of Subclause 16.1, "<embedded SQL host program>"):

   a) Any optional CHARACTER SET specification shall be removed from a <C VARCHAR variable>, a <C character variable>, a <C CLOB variable>, a <C NCHAR variable>, <C NCHAR VARYING variable>, or a <C NCLOB variable>.

   b) The syntax "VARCHAR" shall be replaced by "char" in any <C VARCHAR variable>.

   c) The syntax "BIT" shall be replaced by "char" in any <C bit variable>.

   d) The <length> specified in a <C array specification> in any <C bit variable> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   e) The <length> specified in a <C array specification> in any <C character variable> whose <C character type> specifies "char" or "unsigned char", in any <C VARCHAR variable>, in any <C NCHAR variable>, or in any <C NCHAR VARYING variable>, and the <large object length> specified in a <C CLOB variable> that contains a CHARACTER SET specification or <C NCLOB variable> shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <C host identifier> specified in the containing <C variable definition>.

NOTE 27 – The <length> does not have to be adjusted for <C character type>s that specify "`unsigned short`" because the units of <length> are already the same units as used by the underlying character set.

f) The syntax "`NCHAR`" in any <C NCHAR variable> and the syntax "`NCHAR VARYING `" in any <C NCHAR VARYING variable> shall be replaced by "char".

g) The syntax

```
SQL TYPE IS NCLOB ( L )
```

for a given <C host identifier> *hvn* shall be replaced by

```
struct {
  long          hvn_reserved;
  unsigned long hvn_length;
  char          hvn_data[L];
  } hvn
```

in any <CNCLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

h) The syntax

```
SQL TYPE IS CLOB ( L )
```

or the syntax

```
SQL TYPE IS BLOB ( L )
```

for a given <C host identifier> *hvn* shall be replaced by:

```
struct {
  long          hvn_reserved;
  unsigned long hvn_length;
  char          hvn_data[L];
  } hvn
```

in any <C CLOB variable> or <C BLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

i) The syntax

```
SQL TYPE IS UDTN AS PDT
```

shall be replaced by

```
ADT
```

in any <C user-defined type variable>, where *ADT* is the data type listed in the "C data type" column corresponding to the row for SQL data type *PDT* in Table 19, "Data type correspondences for C", in ISO/IEC 9075-2. *ADT* shall not be "none". The data type identified by *UDTN* is called the *associated user-defined type* of the host variable and the data type identified by *PDT* is called the *associated SQL data type* of the host variable.

j) The syntax

```
SQL TYPE IS BLOB AS LOCATOR
```

shall be replaced by

```
unsigned long
```

in any <C BLOB locator variable>. The host variable defined by <C BLOB locator variable> is called a *binary large object locator variable*.

k) The syntax

```
SQL TYPE IS CLOB AS LOCATOR
```

shall be replaced by

```
unsigned long
```

in any <C CLOB locator variable>. The host variable defined by <C CLOB locator variable> is called a *character large object locator variable*.

l) The syntax

```
SQL TYPE IS <collection type> AS LOCATOR
```

shall be replaced by

```
unsigned long
```

in any <C array locator variable>. The host variable defined by <C array locator variable> is called an *array locator variable*. The data type identified by <collection type> is called the *associated array type* of the host variable.

m) The syntax

```
SQL TYPE IS <user-defined type> AS LOCATOR
```

shall be replaced by

```
unsigned long
```

in any <C user-defined type locator variable>. The host variable defined by <C user-defined type locator variable> is called a *user-defined type locator variable*. The data type identified by <user-defined type> is called the *associated user-defined type* of the host variable.

n) The syntax

```
SQL TYPE IS <reference type>
```

for a given <C host identifier> *hvn* shall be replaced by

```
unsigned char hvn[L]
```

in any <C REF variable>, where *L* is the implementation-defined length in octets of a reference type.

The modified <C variable definition> shall be a valid C data declaration in the program derived from the <embedded SQL C program>.

7) The reference type identified by <reference type> contained in a <C REF variable> is called the *referenced type* of the reference.

8) The <character representation> sequence contained in a <C initial value> specifies an initial value to be assigned to the C variable. It shall be a valid C specification of an initial value.

9) Except for array specifications for character strings and bit strings, a <C variable definition> shall specify a scalar type.

10) In a <C variable definition>, the words “VARCHAR”, “CHARACTER”, “SET”, “IS”, “BIT”, “VARYING”, “BLOB”, “CLOB”, “NCHAR”, “NCLOB”, “AS”, “LOCATOR”, and “REF” may be specified in any combination of upper-case and lower-case letters (see the Syntax Rules of Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2).

11) In a <C character variable>, a <C VARCHAR variable>, or a <C CLOB variable>, if a <character set specification> is specified, then the equivalent SQL data type is CHARACTER, CHARACTER VARYING, or CHARACTER LARGE OBJECT whose character repertoire is the same as the repertoire specified by the <character set specification>. In a <C NCHAR variable>, a <C NCHAR VARYING variable>, or a <C NCLOB variable>, if a <character set specification> is specified, then the equivalent SQL data type is NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, or NATIONAL CHARACTER LARGE OBJECT whose character repertoire is the same as the repertoire specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

12) Each <C host identifier> specified in a <C character variable> or a <C NCHAR variable> describes a fixed-length character string. The length is specified by the <length> of the <C array specification>. The value in the host variable is terminated by a null character and the position occupied by this null character is included in the length of the host variable. The equivalent SQL data type is CHARACTER or NATIONAL CHARACTER, respectively, whose length is one less than the <length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1 (one).

13) Each <C host identifier> specified in a <C VARCHAR variable> or a <C NCHAR VARYING variable> describes a variable-length character string. The maximum length is specified by the <length> of the <C array specification>. The value in the host variable is terminated by a null character and the position occupied by this null character is included in the maximum length of the host variable. The equivalent SQL data type is CHARACTER VARYING or NATIONAL CHARACTER VARYING, respectively, whose maximum length is 1 (one) less than the <length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1 (one).

14) Each <C host identifier> specified in a <C bit variable> describes a fixed-length bit string. The value in the host variable has a BIT_LENGTH of <length>. Let B be the number of bits in a C **char** and let L be the <length> of the <C array specification>. The length of an equivalent C **char** variable is the smallest integer that is not less than the result of L/B. The equivalent SQL data type is BIT whose length is L.

15) “long” describes an exact numeric variable. The equivalent SQL data type is INTEGER or BOOLEAN.

16) “short” describes an exact numeric variable. The equivalent SQL data type is SMALLINT.

17) “float” describes an approximate numeric variable. The equivalent SQL data type is REAL.

18) “double” describes an approximate numeric variable. The equivalent SQL data type is DOUBLE PRECISION.

Access Rules

None.

General Rules

1) See Subclause 16.1, "<embedded SQL host program>".

Conformance Rules

1) Without Feature B012, "Embedded C", <embedded SQL C program> shall not be specified.

2) Without Feature F511, "BIT data type", a <C derived variable> shall not be a <C bit variable>.

3) Without Feature F451, "Character set definition", or Feature F461, "Named character sets", a <C variable definition> shall not contain a <character set specification>.

4) Without Feature S041, "Basic reference types", a <C derived variable> shall not be a <C REF variable>.

5) Without Feature S241, "Transform functions", a <C derived type specification> shall not be a <C user-defined type variable>.

6) Without Feature S232, "Array locators", a <C derived variable> shall not be a <C array locator variable>.

7) Without Feature S231, "Structured type locators", the <user-defined type name> simply contained in a <C user-defined type locator variable> shall not identify a structured type.

8) Without Feature T041, "Basic LOB data type support", a <C BLOB variable>, <C CLOB variable>, <C BLOB locator variable>, and <C CLOB locator variable> shall not be specified.

16.5 <embedded SQL COBOL program>

Function

Specify an <embedded SQL COBOL program>.

Format

```
<embedded SQL COBOL program> ::= !! See the Syntax Rules.

<COBOL variable definition> ::=
      {01|77} <COBOL host identifier> <COBOL type specification>
        [ <character representation>... ] <period>

<COBOL host identifier> ::= !! See the Syntax Rules.

<COBOL type specification> ::=
        <COBOL character type>
      | <COBOL national character type>
      | <COBOL bit type>
      | <COBOL numeric type>
      | <COBOL integer type>
      | <COBOL derived type specification>

<COBOL derived type specification> ::=
        <COBOL CLOB variable>
      | <COBOL NCLOB variable>
      | <COBOL BLOB variable>
      | <COBOL user-defined type variable>
      | <COBOL CLOB locator variable>
      | <COBOL BLOB locator variable>
      | <COBOL array locator variable>
      | <COBOL user-defined type locator variable>
      | <COBOL REF variable>

<COBOL character type> ::=
      [ CHARACTER SET [ IS ] <character set specification> ]
      { PIC | PICTURE } [ IS ] { X [ <left paren> <length> <right paren> ] }...

<COBOL national character type> ::=
      [ CHARACTER SET [ IS ] <character set specification> ]
      { PIC | PICTURE } [ IS ] { N [ <left paren> <length> <right paren> ] }...

<COBOL CLOB variable> ::=
      [ USAGE [ IS ] ]
        SQL TYPE IS CLOB <left paren> <large object length> <right paren>
        [ CHARACTER SET [ IS ] <character set specification> ]

<COBOL NCLOB variable> ::=
      [ USAGE [ IS ] ]
        SQL TYPE IS NCLOB <left paren> <large object length> <right paren>
        [ CHARACTER SET [ IS ] <character set specification> ]

<COBOL bit type> ::=
      { PIC | PICTURE } [ IS ] { X [ <left paren> <length> <right paren> ] }...
        USAGE [ IS ] BIT

<COBOL BLOB variable> ::=
      [ USAGE [ IS ] ]
```

```
        SQL TYPE IS BLOB <left paren> <large object length> <right paren>

<COBOL user-defined type variable> ::=
      [ USAGE [ IS ] ]
        SQL TYPE IS <user-defined type> AS <predefined type>

<COBOL CLOB locator variable> ::=
      [ USAGE [ IS ] ]
        SQL TYPE IS CLOB AS LOCATOR

<COBOL BLOB locator variable> ::=
      [ USAGE [ IS ] ]
        SQL TYPE IS BLOB AS LOCATOR

<COBOL array locator variable> ::=
      [ USAGE [ IS ] ]
        SQL TYPE IS <collection type> AS LOCATOR

<COBOL user-defined type locator variable> ::=
      [ USAGE [ IS ] ]
        SQL TYPE IS <user-defined type name> AS LOCATOR

<COBOL REF variable> ::=
        [ USAGE [ IS ] ]
        SQL TYPE IS <reference type>

<COBOL numeric type> ::=
      { PIC | PICTURE } [ IS ]
        S <COBOL nines specification>
      [ USAGE [ IS ] ] DISPLAY SIGN LEADING SEPARATE

<COBOL nines specification> ::=
        <COBOL nines> [ V [ <COBOL nines> ] ]
      | V <COBOL nines>

<COBOL integer type> ::=
         <COBOL binary integer>

<COBOL binary integer> ::=
      { PIC | PICTURE } [ IS ] S<COBOL nines>
        [ USAGE [ IS ] ] BINARY

<COBOL nines> ::= { 9 [ <left paren> <length> <right paren> ] }...
```

NOTE 28 – The syntax “N(*L*)” is not part of the current COBOL standard, so its use is merely a recommendation; therefore, the production <COBOL national character type> is not normative in ISO/IEC 9075.

Syntax Rules

1) An <embedded SQL COBOL program> is a compilation unit that consists of COBOL text and SQL text. The COBOL text shall conform to the COBOL standard ISO 1989. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> in an <embedded SQL COBOL program> may be specified wherever a COBOL statement may be specified in the Procedure Division of the <embedded SQL COBOL program>. If the COBOL statement could be immediately preceded by a paragraph-name, then the <embedded SQL statement> may be immediately preceded by a paragraph-name.

3) A <COBOL host identifier> is any valid COBOL data-name. A <COBOL host identifier> shall be contained in an <embedded SQL COBOL program>.

4) A <COBOL variable definition> is a restricted form of COBOL data description entry that defines a host variable.

5) The <collection type> contained in a <COBOL array locator variable> shall specify an array data type. The array data type specified by the <collection type> contained in a <COBOL array locator variable> is called the *associated array data type* of the array locator.

6) A <COBOL variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL COBOL program> (see the Syntax Rules of Subclause 16.1, "<embedded SQL host program>").

   a) Any optional CHARACTER SET specification shall be removed from a <COBOL character type>, a <COBOL national character type>, a <COBOL CLOB variable>, and a <COBOL NCLOB variable>.

   b) The syntax

      ```
      USAGE IS BIT
      ```

      shall be deleted.

   c) The <length> specified in any <COBOL bit type> shall be replaced by a length equal to the smallest integer not less than L/B, as defined in the Syntax Rules of this Subclause.

   d) The <length> specified in any <COBOL character type> and the <large object length> specified in any <COBOL CLOB variable> or <COBOL NCLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of PN, where PN is the <COBOL host identifier> specified in the containing <COBOL variable definition>.

      NOTE 29 – The <length> specified in a <COBOL national character type> does not have to be adjusted, because the units of <length> are already the same units as used by the underlying character set.

      NOTE 30 – The syntax "N(L)" is not part of the current COBOL standard, so its use is merely a recommendation; therefore, the production <COBOL national character type> is not normative in ISO/IEC 9075.

   e) The syntax

      ```
      SQL TYPE IS CLOB ( L )
      ```

      or the syntax

      ```
      SQL TYPE IS NCLOB ( L )
      ```

      or the syntax

      ```
      SQL TYPE IS BLOB ( L )
      ```

for a given <COBOL host identifier> *HVN* shall be replaced by:

```
49 HVN-RESERVED PIC S9(9) USAGE IS BINARY.
49 HVN-LENGTH PIC S9(9) USAGE IS BINARY.
49 HVN-DATA PIC X(L).
```

in any <COBOL CLOB variable> or <COBOL BLOB variable>.

f) The syntax

```
SQL TYPE IS UDTN AS PDT
```

shall be replaced by

```
ADT
```

in any <COBOL user-defined type variable>, where *ADT* is the data type listed in the "COBOL data type" column corresponding to the row for SQL data type *PDT* in Table 20, "Data type correspondences for COBOL", in ISO/IEC 9075-2. *ADT* shall not be "none". The data type identified by *UDTN* is called the *associated user-defined type* of the host variable and the data type identified by *PDT* is called the *associated SQL data type* of the host variable.

g) The syntax

```
SQL TYPE IS BLOB AS LOCATOR
```

shall be replaced by

```
PIC S9(9) USAGE IS BINARY
```

in any <COBOL BLOB locator variable>. The host variable defined by <COBOL BLOB locator variable> is called a *binary large object locator variable*.

h) The syntax

```
SQL TYPE IS CLOB AS LOCATOR
```

shall be replaced by

```
PIC S9(9) USAGE IS BINARY
```

in any <COBOL CLOB locator variable>. The host variable defined by <COBOL CLOB locator variable> is called a *character large object locator variable*.

i) The syntax

```
SQL TYPE IS <collection type> AS LOCATOR
```

shall be replaced by

```
PIC S9(9) USAGE IS BINARY
```

in any <COBOL array locator variable>. The host variable defined by <COBOL array locator variable> is called an *array locator variable*. The data type identified by <collection type> is called the *associated array type* of the host variable.

j) The syntax

```
SQL TYPE IS <user-defined type> AS LOCATOR
```

shall be replaced by

```
PIC S9(9) USAGE IS BINARY
```

in any <COBOL user-defined type locator variable>. The host variable defined by <COBOL user-defined type locator variable> is called a *user-defined type locator variable*. The data type identified by <user-defined type> is called the *associated user-defined type* of the host variable.

k) The syntax

```
SQL TYPE IS <reference type>
```

for a given <COBOL host identifier> *HVN* shall be replaced by

```
01 HVN PICTURE X(L)
```

in any <COBOL REF variable>, where *L* is the implementation-defined length in octets of a reference type.

The modified <COBOL variable definition> shall be a valid data description entry in the Data Division of the program derived from the <embedded SQL COBOL program>.

7) The reference type identified by <reference type> contained in a <COBOL REF variable> is called the *referenced type* of the reference.

8) The optional <character representation> sequence in a <COBOL variable definition> may specify a VALUE clause. Whether other clauses may be specified is implementation-defined. The <character representation> sequence shall be such that the <COBOL variable definition> is a valid COBOL data description entry.

9) A <COBOL character type> describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

10) A <COBOL bit type> describes a bit string variable. Let *B* be the number of bits in a COBOL character and let *L* be the <length> of the <COBOL bit type>. The length of an equivalent COBOL character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the <COBOL bit type>. If the length of the <COBOL bit type> is 1 (one), then the equivalent SQL data type may be BOOLEAN.

11) A <COBOL numeric type> describes an exact numeric variable. The equivalent SQL data type is NUMERIC of the same precision and scale.

12) A <COBOL binary integer> describes an exact numeric variable. The equivalent SQL data type is SMALLINT or INTEGER.

Access Rules

None.

General Rules

1) See Subclause 16.1, "<embedded SQL host program>".

Conformance Rules

1) Without Feature B013, “Embedded COBOL”, <embedded SQL COBOL program> shall not be specified.

2) Without Feature F511, “BIT data type”, a <COBOL type specification> shall not be a <COBOL bit type>.

3) Without Feature F451, “Character set definition”, or Feature F461, “Named character sets”, a <COBOL character type> shall not contain a <character set specification>.

4) Without Feature S041, “Basic reference types”, a <COBOL type specification> shall not be a <COBOL REF variable>.

5) Without Feature S241, “Transform functions”, a <COBOL derived type specification> shall not be a <COBOL user-defined type variable>.

6) Without Feature S232, “Array locators”, a <COBOL derived type specification> shall not be a <COBOL array locator variable>.

7) Without Feature S231, “Structured type locators”, the <user-defined type name> simply contained in a <COBOL user-defined type locator variable> shall not identify a structured type.

8) Without Feature T041, “Basic LOB data type support”, a <COBOL BLOB variable>, <COBOL CLOB variable>, <COBOL BLOB locator variable>, and <COBOL CLOB locator variable> shall not be specified.

16.6 <embedded SQL Fortran program>

Function

Specify an <embedded SQL Fortran program>.

Format

```
<embedded SQL Fortran program> ::= !! See the Syntax Rules.

<Fortran variable definition> ::=
      <Fortran type specification>
      <Fortran host identifier> [ { <comma> <Fortran host identifier> }... ]

<Fortran host identifier> ::= !! See the Syntax Rules.

<Fortran type specification> ::=
        CHARACTER [ <asterisk> <length> ]
            [ CHARACTER SET [ IS ] <character set specification> ]
      | CHARACTER KIND = n [ <asterisk> <length> ]
            [ CHARACTER SET [ IS ] <character set specification> ]
      | BIT [ <asterisk> <length> ]
      | INTEGER
      | REAL
      | DOUBLE PRECISION
      | LOGICAL
      | <Fortran derived type specification>

<Fortran derived type specification> ::=
        <Fortran CLOB variable>
      | <Fortran BLOB variable>
      | <Fortran user-defined type variable>
      | <Fortran CLOB locator variable>
      | <Fortran BLOB locator variable>
      | <Fortran user-defined type locator variable>
      | <Fortran array locator variable>
      | <Fortran REF variable>

<Fortran CLOB variable> ::=
      SQL TYPE IS CLOB <left paren> <large object length> <right paren>
        [ CHARACTER SET [ IS ] <character set specification> ]

<Fortran BLOB variable> ::=
      SQL TYPE IS BLOB <left paren> <large object length> <right paren>

<Fortran user-defined type variable> ::=
      SQL TYPE IS <user-defined type> AS <predefined type>

<Fortran CLOB locator variable> ::=
      SQL TYPE IS CLOB AS LOCATOR

<Fortran BLOB locator variable> ::=
      SQL TYPE IS BLOB AS LOCATOR

<Fortran user-defined type locator variable> ::=
      SQL TYPE IS <user-defined type name> AS LOCATOR

<Fortran array locator variable> ::=
      SQL TYPE IS <collection type> AS LOCATOR
```

```
<Fortran REF variable> ::=
      SQL TYPE IS <reference type>
```

Syntax Rules

1) An <embedded SQL Fortran program> is a compilation unit that consists of Fortran text and SQL text. The Fortran text shall conform to the Fortran standard ISO 1539. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever an executable Fortran statement may be specified. An <embedded SQL statement> that precedes any executable Fortran statement in the containing <embedded SQL Fortran program> shall not have a Fortran statement number. Otherwise, if the Fortran statement could have a statement number then the <embedded SQL statement> can have a statement number.

3) Blanks are significant in <embedded SQL statement>s. The rules for <separator>s in an <embedded SQL statement> are as specified in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

4) A <Fortran host identifier> is any valid Fortran variable name with all <space> characters removed. A <Fortran host identifier> shall be contained in an <embedded SQL Fortran program>.

5) A <Fortran variable definition> is a restricted form of Fortran type-statement that defines one or more host variables.

6) A <Fortran variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Fortran program> (see the Syntax Rules Subclause 16.1, “<embedded SQL host program>”).

   a) Any optional CHARACTER SET specification shall be removed from the CHARACTER and the CHARACTER KIND=*n* alternatives in a <Fortran type specification>.

   b) The <length> specified in the CHARACTER alternative of any <Fortran type specification> and the <large object length> specified in any <Fortran CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Fortran host identifier> specified in the containing <Fortran variable definition>.

      NOTE 31 – The <length> does not have to be adjusted for CHARACTER KIND=*n* alternatives of any <Fortran type specification>, because the units of <length> are already the same units as used by the underlying character set.

   c) The syntax “BIT” shall be replaced by “CHARACTER” in any BIT alternative of a <Fortran type specification>.

   d) The <length> specified in any BIT alternative of a <Fortran type specification> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   e) The syntax

      ```
      SQL TYPE IS CLOB ( L )
      ```

and the syntax

```
SQL TYPE IS BLOB ( L )
```

for a given <Fortran host identifier> *HVN* shall be replaced by

```
INTEGER HVN_RESERVED
INTEGER HVN_LENGTH
CHARACTER HVN_DATA [ <asterisk> L ]
```

in any <Fortran CLOB variable> or <Fortran BLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.2, "<token> and <separator>", of ISO/IEC 9075-2.

f) The syntax

```
SQL TYPE IS UDTN AS PDT
```

shall be replaced by

```
ADT
```

in any <Fortran user-defined type variable>, where *ADT* is the data type listed in the “Fortran data type” column corresponding to the row for SQL data type *PDT* in Table 21, "Data type correspondences for Fortran", in ISO/IEC 9075-2. *ADT* shall not be “none”. The data type identified by *UDTN* is called the *associated user-defined type* of the host variable and the data type identified by *PDT* is called the *associated SQL data type* of the host variable.

g) The syntax

```
SQL TYPE IS BLOB AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Fortran BLOB locator variable>. The host variable defined by <Fortran BLOB locator variable> is called a *binary large object locator variable*.

h) The syntax

```
SQL TYPE IS CLOB AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Fortran CLOB locator variable>. The host variable defined by <Fortran CLOB locator variable> is called a *character large object locator variable*.

i) The syntax

```
SQL TYPE IS <user-defined type> AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Fortran user-defined type locator variable>. The host variable defined by <Fortran user-defined type locator variable> is called a *user-defined type locator variable*. The data type identified by <user-defined type> is called the *associated user-defined type* of the host variable.

j) The syntax

```
SQL TYPE IS <collection type> AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Fortran array locator variable>. The host variable defined by <Fortran array locator variable> is called an *array locator variable*. The data type identified by <collection type> is called the *associated array type* of the host variable.

k) The syntax

```
SQL TYPE IS <reference type>
```

for a given <Fortran host identifier> *HVN* shall be replaced by

```
CHARACTER HVN * <length>
```

in any <Fortran reference type>, where <length> is the implementation-defined in octets of a reference type.

The modified <Fortran variable definition> shall be a valid Fortran type-statement in the program derived from the <embedded SQL Fortran program>.

7) The reference type identified by <reference type> contained in an <Fortran REF variable> is called the *referenced type* of the reference.

8) CHARACTER without "KIND=*n*" describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

9) CHARACTER KIND=*n* describes a character string variable whose equivalent SQL data type is either CHARACTER or NATIONAL CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit. The value of *n* determines implementation-defined characteristics of the Fortran variable; values of *n* are implementation-defined.

10) BIT describes a bit string variable. Let *B* be the number of bits in a Fortran character and let *L* be the <length> of the bit string variable. The length of an equivalent Fortran character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the bit string variable.

11) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

12) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

13) DOUBLE PRECISION describes an approximate numeric variable. The equivalent SQL data type is DOUBLE PRECISION.

14) LOGICAL describes a boolean variable. The equivalent SQL data type is BOOLEAN.

Access Rules

None.

General Rules

1) See Subclause 16.1, "<embedded SQL host program>".

Conformance Rules

1) Without Feature B014, "Embedded Fortran", <embedded SQL Fortran program> shall not be specified.

2) Without Feature F511, "BIT data type", a <Fortran type specification> shall not specify BIT.

3) Without Feature F451, "Character set definition", or Feature F461, "Named character sets", a <Fortran type specification> shall not contain a <character set specification>.

4) Without Feature S041, "Basic reference types", a <Fortran derived type specification> shall not be a <Fortran REF variable>.

5) Without Feature S241, "Transform functions", a <Fortran derived type specification> shall not be a <Fortran user-defined type variable>.

6) Without Feature S232, "Array locators", a <Fortran derived type specification> shall not be a <Fortran array locator variable>.

7) Without Feature S231, "Structured type locators", the <user-defined type name> simply contained in a <Fortran user-defined type locator variable> shall not identify a structured type.

8) Without Feature T041, "Basic LOB data type support", a <Fortran BLOB variable>, <Fortran CLOB variable>, <Fortran BLOB locator variable>, and <Fortran CLOB locator variable> shall not be specified.

16.7 <embedded SQL MUMPS program>

Function

Specify an <embedded SQL MUMPS program>.

Format

```
<embedded SQL MUMPS program> ::= !! See the Syntax Rules.

<MUMPS variable definition> ::=
          <MUMPS numeric variable> <semicolon>
        | <MUMPS character variable> <semicolon>
        | <MUMPS derived type specification> <semicolon>

<MUMPS character variable> ::=
        VARCHAR <MUMPS host identifier> <MUMPS length specification>
          [ { <comma> <MUMPS host identifier> <MUMPS length specification> }... ]

<MUMPS host identifier> ::= !! See the Syntax Rules.

<MUMPS length specification> ::=
        <left paren> <length> <right paren>

<MUMPS numeric variable> ::=
        <MUMPS type specification>
          <MUMPS host identifier> [ { <comma> <MUMPS host identifier> }... ]

<MUMPS type specification> ::=
          INT
        | DEC [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
        | REAL

<MUMPS derived type specification> ::=
          <MUMPS CLOB variable>
        | <MUMPS BLOB variable>
        | <MUMPS user-defined type variable>
        | <MUMPS CLOB locator variable>
        | <MUMPS BLOB locator variable>
        | <MUMPS user-defined type locator variable>
        | <MUMPS array locator variable>
        | <MUMPS REF variable>

<MUMPS CLOB variable> ::=
        SQL TYPE IS CLOB <left paren> <large object length> <right paren>
          [ CHARACTER SET [ IS ] <character set specification> ]


<MUMPS BLOB variable> ::=
        SQL TYPE IS BLOB <left paren> <large object length> <right paren>

<MUMPS user-defined type variable> ::=
        SQL TYPE IS <user-defined type> AS <predefined type>

<MUMPS CLOB locator variable> ::=
        SQL TYPE IS CLOB AS LOCATOR


<MUMPS BLOB locator variable> ::=
      SQL TYPE IS BLOB AS LOCATOR


<MUMPS user-defined type locator variable> ::=
      SQL TYPE IS <user-defined type name> AS LOCATOR

<MUMPS array locator variable> ::=
      SQL TYPE IS <collection type> AS LOCATOR

<MUMPS REF variable> ::=
      SQL TYPE IS <reference type>
```

Syntax Rules

1) An <embedded SQL MUMPS program> is a compilation unit that consists of MUMPS text and SQL text. The MUMPS text shall conform to the MUMPS standard ISO/IEC 11756. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) A <MUMPS host identifier> is any valid MUMPS variable name. A <MUMPS host identifier> shall be contained in an <embedded SQL MUMPS program>.

3) An <embedded SQL statement> may be specified wherever a MUMPS command may be specified.

4) A <MUMPS variable definition> defines one or more host variables.

5) The <MUMPS character variable> describes a variable-length character string. The equivalent SQL data type is CHARACTER VARYING whose maximum length is the <length> of the <MUMPS length specification> and whose character set is implementation-defined.

6) INT describes an exact numeric variable. The equivalent SQL data type is INTEGER.

7) DEC describes an exact numeric variable. The <scale> shall not be greater than the <precision>. The equivalent SQL data type is DECIMAL with the same <precision> and <scale>.

8) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

9) A <MUMPS derived type specification> shall be modified as follows before it is placed into the program derived from the <embedded SQL MUMPS program> (see the Syntax Rules of Subclause 16.1, "<embedded SQL host program>").

   a) Any optional CHARACTER SET specification shall be removed from a <MUMPS CLOB variable>.

   b) The syntax

      `SQL TYPE IS CLOB ( L )`

      and the syntax

      `SQL TYPE IS BLOB ( L )`

for a given <MUMPS host identifier> *HVN* shall be replaced by

```
INT HVN_RESERVED
INT HVN_LENGTH
VARCHAR HVN_DATA L
```

in any <MUMPS CLOB variable> or <MUMPS BLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.2, "<token> and <separator>", of ISO/IEC 9075-2.

c) The syntax

```
SQL TYPE IS UDTN AS PDT
```

shall be replaced by

```
ADT
```

in any <MUMPS user-defined type variable>, where *ADT* is the data type listed in the "MUMPS data type" column corresponding to the row for SQL data type *PDT* in Table 22, "Data type correspondences for MUMPS", in ISO/IEC 9075-2. *ADT* shall not be "none". The data type identified by *UDTN* is called the *associated user-defined type* of the host variable and the data type identified by *PDT* is called the *associated SQL data type* of the host variable.

d) The syntax

```
SQL TYPE IS BLOB AS LOCATOR
```

shall be replaced by

```
INT
```

in any or <MUMPS BLOB locator variable>. The host variable defined by <MUMPS BLOB locator variable> is called a *binary large object locator variable*.

e) The syntax

```
SQL TYPE IS CLOB AS LOCATOR
```

shall be replaced by

```
INT
```

in any <MUMPS CLOB locator variable>. The host variable defined by <MUMPS CLOB locator variable> is called a *character large object locator variable*.

f) The syntax

```
SQL TYPE IS <user-defined type> AS LOCATOR
```

shall be replaced by

```
INT
```

in any <MUMPS user-defined type locator variable>. The host variable defined by <MUMPS user-defined type locator variable> is called a *user-defined type locator variable*. The data type identified by <user-defined type> is called the *associated user-defined type* of the host variable.

g) The syntax

```
SQL TYPE IS <collection type> AS LOCATOR
```

shall be replaced by

```
INT
```

in any <MUMPS array locator variable>. The host variable defined by <MUMPS array locator variable> is called an *array locator variable*. The data type identified by <collection type> is called the *associated array type* of the host variable.

h) The syntax

```
SQL TYPE IS <reference type>
```

for a given <MUMPS host identifier> *HVN* shall be replaced by

```
VARCHAR HVN L
```

in any <MUMPS reference type>, where *L* is the implementation-defined length in octets of a reference type.

The modified <MUMPS variable definition> shall be a valid MUMPS variable in the program derived from the <embedded SQL MUMPS program>.

10) The reference type identified by <reference type> contained in an <MUMPS REF variable> is called the *referenced type* of the reference.

Access Rules

None.

General Rules

1) See Subclause 16.1, "<embedded SQL host program>".

Conformance Rules

1) Without Feature B015, "Embedded MUMPS", <embedded SQL MUMPS program> shall not be specified.

2) Without Feature S041, "Basic reference types", a <MUMPS derived type specification> shall not be a <MUMPS REF variable>.

3) Without Feature S241, "Transform functions", a <MUMPS derived type specification> shall not be a <MUMPS user-defined type variable>.

4) Without Feature S232, "Array locators", a <MUMPS derived type specification> shall not be a <MUMPS array locator variable>.

5) Without Feature S231, "Structured type locators", the <user-defined type name> simply contained in a <MUMPS user-defined type locator variable> shall not identify a structured type.

6) Without Feature T041, "Basic LOB data type support", a <MUMPS BLOB variable>, <MUMPS CLOB variable>, <MUMPS BLOB locator variable>, and <MUMPS CLOB locator variable> shall not be specified.

16.8 <embedded SQL Pascal program>

Function

Specify an <embedded SQL Pascal program>.

Format

```
<embedded SQL Pascal program> ::= !! See the Syntax Rules.

<Pascal variable definition> ::=
       <Pascal host identifier> [ { <comma> <Pascal host identifier> }... ] <colon>
         <Pascal type specification> <semicolon>

<Pascal host identifier> ::= !! See the Syntax Rules.

<Pascal type specification> ::=
         PACKED ARRAY <left bracket> 1 <double period> <length> <right bracket>
           OF CHAR
             [ CHARACTER SET [ IS ] <character set specification> ]
       | PACKED ARRAY <left bracket> 1 <double period> <length> <right bracket>
           OF BIT
       | INTEGER
       | REAL
       | CHAR [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT
       | BOOLEAN
       | <Pascal derived type specification>

<Pascal derived type specification> ::=
         <Pascal CLOB variable>
       | <Pascal BLOB variable>
       | <Pascal user-defined type variable>
       | <Pascal CLOB locator variable>
       | <Pascal BLOB locator variable>
       | <Pascal user-defined type locator variable>
       | <Pascal array locator variable>
       | <Pascal REF variable>

<Pascal CLOB variable> ::=
       SQL TYPE IS CLOB <left paren> <large object length> <right paren>
         [ CHARACTER SET [ IS ] <character set specification> ]

<Pascal BLOB variable> ::=
       SQL TYPE IS BLOB <left paren> <large object length> <right paren>

<Pascal CLOB locator variable> ::=
       SQL TYPE IS CLOB AS LOCATOR

<Pascal user-defined type variable> ::=
       SQL TYPE IS <user-defined type> AS <predefined type>

<Pascal BLOB locator variable> ::=
       SQL TYPE IS BLOB AS LOCATOR

<Pascal user-defined type locator variable> ::=
       SQL TYPE IS <user-defined type name> AS LOCATOR

<Pascal array locator variable> ::=
          SQL TYPE IS <collection type> AS LOCATOR

<Pascal REF variable> ::=
          SQL TYPE IS <reference type>
```

Syntax Rules

1) An <embedded SQL Pascal program> is a compilation unit that consists of Pascal text and SQL text. The Pascal text shall conform to one of the Pascal standards ISO/IEC 7185 and ISO/IEC 10206. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a Pascal statement may be specified. An <embedded SQL statement> may be prefixed by a Pascal label.

3) A <Pascal host identifier> is a Pascal variable-identifier whose applied instance denotes a defining instance within an <embedded SQL begin declare> and an <embedded SQL end declare>.

4) A <Pascal variable definition> defines one or more <Pascal host identifier>s.

5) A <Pascal variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Pascal program> (see the Syntax Rules of Subclause 16.1, "<embedded SQL host program>").

   a) Any optional CHARACTER SET specification shall be removed from the PACKED ARRAY OF CHAR or CHAR alternatives of a <Pascal type specification> and a <Pascal CLOB variable>.

   b) The <length> specified in the PACKED ARRAY OF CHAR alternative of any <Pascal type specification> that contains a CHARACTER SET specification and the <large object length> specified in a <Pascal CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Pascal host identifier> specified in the containing <Pascal variable definition>.

   c) If any <Pascal type specification> specifies the syntax "CHAR" and contains a CHARACTER SET specification, then let *L* be a length equal to the length in octets of *PN* and *PN* be the <Pascal host identifier> specified in the containing <Pascal variable definition>. If *L* is greater than 1 (one), then "CHAR" shall be replaced by "PACKED ARRAY [1..*L*] OF CHAR".

   d) The syntax "BIT" shall be replaced by "CHAR" in any PACKED ARRAY OF BIT or BIT alternatives of a <Pascal type specification>.

   e) The <length> specified in any PACKED ARRAY OF BIT alternative in a <Pascal type specification> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   f) The syntax

      ```
      SQL TYPE IS CLOB ( L )
      ```

      and the syntax

      ```
      SQL TYPE IS BLOB ( L )
      ```

for a given <Pascal host identifier> *HVN* shall be replaced by

```
VAR HVN = RECORD
  HVN_RESERVED : INTEGER;
  HVN_LENGTH : INTEGER;
  HVN_DATA : PACKED ARRAY [ 1..L ] OF CHAR;
END;
```

in any <Pascal CLOB variable> or <Pascal BLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.2, "<token> and <separator>", of ISO/IEC 9075-2.

g) The syntax

```
SQL TYPE IS UDTN AS PDT
```

shall be replaced by

```
ADT
```

in any <Pascal user-defined type variable>, where *ADT* is the data type listed in the “Pascal data type” column corresponding to the row for SQL data type *PDT* in Table 23, "Data type correspondences for Pascal", in ISO/IEC 9075-2. *ADT* shall not be “none”. The data type identified by *UDTN* is called the *associated user-defined type* of the host variable and the data type identified by *PDT* is called the *associated SQL data type* of the host variable.

h) The syntax

```
SQL TYPE IS BLOB AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Pascal BLOB locator variable>. The host variable defined by <Pascal BLOB locator variable> is called a *binary large object locator variable*.

i) The syntax

```
SQL TYPE IS CLOB AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Pascal CLOB locator variable>. The host variable defined by <Pascal CLOB locator variable> is called a *character large object locator variable*.

j) The syntax

```
SQL TYPE IS <user-defined type> AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Pascal user-defined type locator variable>. The host variable defined by <Pascal user-defined type locator variable> is called a *user-defined type locator variable*. The data type identified by <user-defined type> is called the *associated user-defined type* of the host variable.

k) The syntax

```
SQL TYPE IS <collection type> AS LOCATOR
```

shall be replaced by

```
INTEGER
```

in any <Pascal array locator variable>. The host variable defined by <Pascal array locator variable> is called an *array locator variable*. The data type identified by <collection type> is called the *associated array type* of the host variable.

l) The syntax

```
SQL TYPE IS <reference type>
```

for a given <Pascal host identifier> *HVN* shall be replaced by

```
HVN : PACKED ARRAY [1..<length>] OF CHAR
```

in any <Pascal reference type>, where <length> is the implementation-defined length in octets of a reference type.

The modified <Pascal variable definition> shall be a valid Pascal variable-declaration in the program derived from the <embedded SQL Pascal program>.

6) The reference type identified by <reference type> contained in an <Pascal REF variable> is called the *referenced type* of the reference.

7) CHAR specified without a CHARACTER SET specification is the ordinal-type-identifier of PASCAL. The equivalent SQL data type is CHARACTER with length 1 (one).

8) BIT describes a single-bit variable. It is mapped to a Pascal CHAR ordinal-type-identifier whose most significant bit contains either 0 (zero) or 1 (one) and whose least significant bits contain 0 (zero). The equivalent SQL data type is BIT with length 1 (one).

9) PACKED ARRAY [1..<length>] OF CHAR describes a character string having 2 or more components of the simple type CHAR. The equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

10) PACKED ARRAY [1..<length>] OF BIT describes a bit string variable. Let *B* be the number of bits in a Pascal CHAR and let *L* be the <length> of the bit string variable. The length of an equivalent Pascal character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the bit string variable.

11) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

12) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

13) BOOLEAN describes a boolean variable. The equivalent SQL data type is BOOLEAN.

Access Rules

None.

General Rules

1) See Subclause 16.1, "<embedded SQL host program>".

Conformance Rules

1) Without Feature B016, "Embedded Pascal", <embedded SQL Pascal program> shall not be specified.

2) Without Feature F511, "BIT data type", a <Pascal type specification> shall not specify BIT or PACKED ARRAY [1..<length>] OF BIT.

3) Without Feature F451, "Character set definition", or Feature F461, "Named character sets", a <Pascal type specification> shall not contain a <character set specification>.

4) Without Feature S041, "Basic reference types", a <Pascal derived type specification> shall not be a <Pascal REF variable>.

5) Without Feature S241, "Transform functions", a <Pascal derived type specification> shall not be a <Pascal user-defined type variable>.

6) Without Feature S232, "Array locators", a <Pascal derived type specification> shall not be a <Pascal array locator variable>.

7) Without Feature S231, "Structured type locators", the <user-defined type name> simply contained in a <Pascal user-defined type locator variable> shall not identify a structured type.

8) Without Feature T041, "Basic LOB data type support", a <Pascal BLOB variable>, <Pascal CLOB variable>, <Pascal BLOB locator variable>, and <Pascal CLOB locator variable> shall not be specified.

16.9 <embedded SQL PL/I program>

Function

Specify an <embedded SQL PL/I program>.

Format

```
<embedded SQL PL/I program> ::= !! See the Syntax Rules.

<PL/I variable definition> ::=
       {DCL | DECLARE}
          {   <PL/I host identifier>
            | <left paren> <PL/I host identifier>
                  [ { <comma> <PL/I host identifier> }... ] <right paren> }
       <PL/I type specification>
       [ <character representation>... ] <semicolon>

<PL/I host identifier> ::= !! See the Syntax Rules.

<PL/I type specification> ::=
         { CHAR | CHARACTER } [ VARYING ] <left paren><length><right paren>
             [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT [ VARYING ] <left paren><length><right paren>
       | <PL/I type fixed decimal> <left paren> <precision>
             [ <comma> <scale> ] <right paren>
       | <PL/I type fixed binary> [ <left paren> <precision> <right paren> ]
       | <PL/I type float binary> <left paren> <precision> <right paren>
       | <PL/I derived type specification>

<PL/I derived type specification> ::=
         <PL/I CLOB variable>
       | <PL/I BLOB variable>
       | <PL/I user-defined type variable>
       | <PL/I CLOB locator variable>
       | <PL/I BLOB locator variable>
       | <PL/I user-defined type locator variable>
       | <PL/I array locator variable>
       | <PL/I REF variable>

<PL/I CLOB variable> ::=
       SQL TYPE IS CLOB <left paren> <large object length> <right paren>
         [ CHARACTER SET [ IS ] <character set specification> ]

<PL/I BLOB variable> ::=
       SQL TYPE IS BLOB <left paren> <large object length> <right paren>

<PL/I user-defined type variable> ::=
       SQL TYPE IS <user-defined type> AS <predefined type>

<PL/I CLOB locator variable> ::=
       SQL TYPE IS CLOB AS LOCATOR

<PL/I BLOB locator variable> ::=
       SQL TYPE IS BLOB AS LOCATOR

<PL/I user-defined type locator variable> ::=
       SQL TYPE IS <user-defined type name> AS LOCATOR
```

```
<PL/I array locator variable> ::=
      SQL TYPE IS <collection type> AS LOCATOR

<PL/I REF variable> ::=
      SQL TYPE IS <reference type>

<PL/I type fixed decimal> ::=
        { DEC | DECIMAL } FIXED
      | FIXED { DEC | DECIMAL }

<PL/I type fixed binary> ::=
        { BIN | BINARY } FIXED
      | FIXED { BIN | BINARY }

<PL/I type float binary> ::=
        { BIN | BINARY } FLOAT
      | FLOAT { BIN | BINARY }
```

Syntax Rules

1) An <embedded SQL PL/I program> is a compilation unit that consists of PL/I text and SQL text. The PL/I text shall conform to the PL/I standard ISO 6160. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a PL/I statement may be specified within a procedure block. If the PL/I statement could include a label prefix, the <embedded SQL statement> may be immediately preceded by a label prefix.

3) A <PL/I host identifier> is any valid PL/I variable identifier. A <PL/I host identifier> shall be contained in an <embedded SQL PL/I program>.

4) A <PL/I variable definition> defines one or more host variables.

5) A <PL/I variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL PL/I program> (see the Syntax Rules of Subclause 16.1, “<embedded SQL host program>”).

   a) Any optional CHARACTER SET specification shall be removed from the CHARACTER or CHARACTER VARYING alternatives of a <PL/I type specification>.

   b) The <length> specified in the CHARACTER or CHARACTER VARYING alternatives of any <PL/I type specification> or a <PL/I CLOB variable> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <PL/I host identifier> specified in the containing <PL/I variable definition>.

   c) The syntax

      SQL TYPE IS CLOB (*L*)

      and the syntax

      SQL TYPE IS BLOB (*L*)

for a given <PL/I host identifier> *HVN* shall be replaced by

```
DCL 1 HVN
  2 HVN_RESERVED FIXED BINARY(31),
  2 HVN_LENGTH FIXED BINARY(31),
  2 HVN_DATA CHARACTER(<length>);
```

in any <PL/I CLOB variable> or <PL/I BLOB variable>, where *L* is the numeric value of <large object length> as specified in Subclause 5.2, "<token> and <separator>", of ISO/IEC 9075-2.

d) The syntax

```
SQL TYPE IS UDTN AS PDT
```

shall be replaced by

```
ADT
```

in any <PL/I user-defined type variable>, where *ADT* is the data type listed in the “PL/I data type” column corresponding to the row for SQL data type *PDT* in Table 24, "Data type correspondences for PL/I", in ISO/IEC 9075-2. *ADT* shall not be “none”. The data type identified by *UDTN* is called the *associated user-defined type* of the host variable and the data type identified by *PDT* is called the *associated SQL data type* of the host variable.

e) The syntax

```
SQL TYPE IS BLOB AS LOCATOR
```

shall be replaced by

```
FIXED BINARY(31)
```

in any <PL/I BLOB locator variable>. The host variable defined by <PL/I BLOB locator variable> is called a *binary large object locator variable*.

f) The syntax

```
SQL TYPE IS CLOB AS LOCATOR
```

shall be replaced by

```
FIXED BINARY(31)
```

in any <PL/I CLOB locator variable>. The host variable defined by <PL/I CLOB locator variable> is called a *character large object locator variable*.

g) The syntax

```
SQL TYPE IS <user-defined type> AS LOCATOR
```

shall be replaced by

```
FIXED BINARY(31)
```

in any <PL/I user-defined type locator variable>. The host variable defined by <PL/I user-defined type locator variable> is called a *user-defined type locator variable*. The data type identified by <user-defined type> is called the *associated user-defined type* of the host variable.

h) The syntax

```
SQL TYPE IS <collection type> AS LOCATOR
```

shall be replaced by

```
FIXED BINARY(31)
```

in any <PL/I array locator variable>. The host variable defined by <PL/I array locator variable> is called an *array locator variable*. The data type identified by <collection type> is called the *associated array type* of the host variable.

i) The syntax

```
SQL TYPE IS <reference type>
```

for a given <PL/I host identifier> *HVN* shall be replaced by

```
DCL HVN CHARACTER(<length>) VARYING
```

in any <PL/I reference type>, where <length> is the implementation-defined length in octets of a reference type.

The modified <PL/I variable definition> shall be a valid PL/I data declaration in the program derived from the <embedded SQL PL/I program>.

6) The reference type identified by <reference type> contained in an <PL/I REF variable> is called the *referenced type* of the reference.

7) A <PL/I variable definition> shall specify a scalar variable, not an array or structure.

8) The optional <character representation> sequence in a <PL/I variable definition> may specify an INITIAL clause. Whether other clauses may be specified is implementation-defined. The <character representation> sequence shall be such that the <PL/I variable definition> is a valid PL/I DECLARE statement.

9) CHARACTER describes a character string variable whose equivalent SQL data type has the character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

Case:

a) If VARYING is not specified, then the length of the variable is fixed. The equivalent SQL data type is CHARACTER with the same length.

b) If VARYING is specified, then the variable is of variable length, with maximum size the value of <length>. The equivalent SQL data type is CHARACTER VARYING with the same maximum length.

10) BIT describes a bit string variable.

Case:

a) If VARYING is not specified, then the length of the variable is fixed. The equivalent SQL data type is BIT with the same length. If the length of the variable is 1 (one), then the equivalent SQL data type may be BOOLEAN.

b) If VARYING is specified, then the variable is of variable length with maximum size of the value of <length>. The equivalent SQL data type is BIT VARYING with the same maximum length.

11) FIXED DECIMAL describes an exact numeric variable. The <scale> shall not be greater than the <precision>. The equivalent SQL data type is DECIMAL with the same <precision> and <scale>.

12) FIXED BINARY describes an exact numeric variable. The equivalent SQL data type is SMALLINT or INTEGER.

13) FLOAT BINARY describes an approximate numeric variable. The equivalent SQL data type is FLOAT with the same <precision>.

Access Rules

None.

General Rules

1) See Subclause 16.1, "<embedded SQL host program>"

Conformance Rules

1) Without Feature B017, "Embedded PL/I", <embedded SQL PL/I program> shall not be specified.

2) Without Feature F511, "BIT data type", a <PL/I type specification> shall not specify BIT or BIT VARYING.

3) Without Feature F451, "Character set definition", or Feature F461, "Named character sets", a <PL/I type specification> shall not contain a <character set specification>.

4) Without Feature S041, "Basic reference types", a <PL/I derived type specification> shall not be a <PL/I REF variable>.

5) Without Feature S241, "Transform functions", a <PL/I derived type specification> shall not be a <PL/I user-defined type variable>.

6) Without Feature S232, "Array locators", a <PL/I derived type specification> shall not be a <PL/I array locator variable>.

7) Without Feature S231, "Structured type locators", the <user-defined type name> simply contained in a <PL/I user-defined type locator variable> shall not identify a structured type.

8) Without Feature T041, "Basic LOB data type support", a <PL/I BLOB variable>, <PL/I CLOB variable>, <PL/I BLOB locator variable>, and <PL/I CLOB locator variable> shall not be specified.

17 Direct invocation of SQL

17.1 <direct SQL statement>

Function

Specify direct execution of SQL.

Format

```
<direct SQL statement> ::=
      <directly executable statement> <semicolon>

<directly executable statement> ::=
        <direct SQL data statement>
      | <SQL schema statement>
      | <SQL transaction statement>
      | <SQL connection statement>
      | <SQL session statement>
      | <direct implementation-defined statement>

<direct SQL data statement> ::=
        <delete statement: searched>
      | <direct select statement: multiple rows>
      | <insert statement>
      | <update statement: searched>
      | <temporary table declaration>

<direct implementation-defined statement> ::= !! See the Syntax Rules
```

Syntax Rules

1) The <direct SQL data statement> shall not contain any SQL parameter reference, SQL variable reference, <dynamic parameter specification>, or <embedded variable specification>.

2) The <value specification> that represents the null value is implementation-defined.

3) The Format and Syntax Rules for <direct implementation-defined statement> are implementation-defined.

Access Rules

1) The Access Rules for <direct implementation-defined statement> are implementation-defined.

General Rules

1) The following <direct SQL statement>s are transaction-initiating <direct SQL statement>s:

   a) <direct SQL statement>s that are transaction-initiating <SQL procedure statement>s;

   b) <direct select statement: multiple rows>; and

   c) <direct implementation-defined statement>s that are transaction-initiating.

2) After the last invocation of an SQL-statement by an SQL-agent in an SQL-session:

   a) A <rollback statement> or a <commit statement> is effectively executed. If an unrecoverable error has occurred, or if the direct invocation of SQL terminated unexpectedly, or if any constraint is not satisfied, then a <rollback statement> is performed. Otherwise, the choice of which of these SQL-statements to perform is implementation-dependent. The determination of whether a direct invocation of SQL has terminated unexpectedly is implementation-dependent.

   b) Let *D* be the <descriptor name> of any system descriptor area that is currently allocated within the current SQL-session. A <deallocate descriptor statement> that specifies

      ```
      DEALLOCATE DESCRIPTOR D
      ```

      is effectively executed.

   c) All SQL-sessions associated with the SQL-agent are terminated.

3) Let *S* be the <direct SQL statement>.

4) The current authorization identifier for privilege determination for the execution of *S* is the SQL-session user identifier.

5) If *S* does not conform to the Format, Syntax Rules, and Access Rules for a <direct SQL statement>, then an exception condition is raised: *syntax error or access rule violation*.

6) When *S* is invoked by the SQL-agent:

   Case:

   a) If *S* is an <SQL connection statement>, then:

      i) The diagnostics area is emptied.

      ii) *S* is executed.

      iii) If *S* successfully initiated or resumed an SQL-session, then subsequent invocations of a <direct SQL statement> by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

   b) Otherwise:

      i) If no SQL-session is current for the SQL-agent, then

         Case:

         1) If the SQL-agent has not executed an <SQL connection statement> and there is no default SQL-session associated with the SQL-agent, then the following <connect statement> is effectively executed:

            ```
            CONNECT TO DEFAULT
            ```

         2) If the SQL-agent has not executed an <SQL connection statement> and there is a default SQL-session associated with the SQL-agent, then the following <set connection statement> is effectively executed:

            ```
            SET CONNECTION DEFAULT
            ```

3) Otherwise, an exception condition is raised: *connection exception — connection does not exist*.

Subsequent calls to an <externally-invoked procedure> or invocations of a <direct SQL statement> by the SQL-agent are associated with the SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

ii) If an SQL-transaction is active for the SQL-agent, then *S* is associated with that SQL-transaction. If *S* is a <direct implementation-defined statement>, then it is implementation-defined whether or not *S* may be associated with an active SQL-transaction; if not, then an exception condition is raised: *invalid transaction state — active SQL-transaction*.

iii) If no SQL-transaction is active for the SQL-agent, then

1) Case:

A) If *S* is a transaction-initiating <direct SQL statement>, then an SQL-transaction is initiated.

B) If *S* is a <direct implementation-defined statement>, then it is implementation-defined whether or not *S* initiates an SQL-transaction. If an implementation defines *S* to be transaction-initiating, then an SQL-transaction is initiated.

2) If *S* initiated an SQL-transaction, then:

A) Let *T* be the SQL-transaction initiated by *S*.

B) *T* is associated with this invocation and any subsequent invocations of <direct SQL statement>s or calls to an <externally-invoked procedure> by the SQL-agent until the SQL-agent terminates *T*.

C) If *S* is not a <start transaction statement>, then

Case:

I) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session (or if there has been no previous SQL-transaction in the SQL-session and a <set transaction statement> has been executed), then the access mode, constraint mode, and isolation level of *T* are set as specified by the <set transaction statement>.

II) Otherwise, the access mode, constraint mode for all constraints, and isolation level for *T* are *read-write*, *immediate*, and SERIALIZABLE, respectively.

D) *T* is associated with the SQL-session.

iv) If *S* contains an <SQL schema statement> and the access mode of the current SQL-transaction is *read-only*, then an exception condition is raised: *invalid transaction state — read-only SQL-transaction*.

v) The diagnostics area is emptied.

vi) *S* is executed.

7) If the execution of a <direct SQL data statement> occurs within the same SQL-transaction as the execution of an SQL-schema statement and this is not allowed by the SQL-implementation, then an exception condition is raised: *invalid transaction state — schema and data statement mixing not supported*.

8) Case:

   a) If *S* executed successfully, then either a completion condition is raised: *successful completion*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*.

   b) If *S* did not execute successfully, then all changes made to SQL-data or schemas by the execution of *S* are canceled and an exception condition is raised.

      NOTE 32 – The method of raising a condition is implementation-defined.

9) Diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Clause 18, “Diagnostics management”.

   NOTE 33 – The method of accessing the diagnostics information is implementation-defined, but does not alter the contents of the diagnostics area.

Conformance Rules

1) Without Feature B021, “Direct SQL”, <direct SQL statement> shall not be specified.

17.2 <direct select statement: multiple rows>

Function

Specify a statement to retrieve multiple rows from a specified table.

Format

```
<direct select statement: multiple rows> ::=
      <query expression> [ <order by clause> ]
```

Syntax Rules

1) All Syntax Rules of Subclause 7.12, "<query expression>", in ISO/IEC 9075-2, apply to the <direct select statement: multiple rows>.

2) The <query expression> or <order by clause> of a <direct select statement: multiple rows> shall not contain any <value specification> other than a <literal>, CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, or CURRENT_PATH.

3) Let *T* be the table specified by the <query expression>.

4) If ORDER BY is specified, then each <sort specification> in the <order by clause> shall contain a <value expression> *VE*, which shall not contain a <subquery> or a <set function specification>, but shall contain a <column reference>.

   a) Let *X* be any <column reference> directly contained in *VE*.

   b) If *X* does not contain an explicit <table or query name> or <correlation name>, then *VE* shall be a <column name> that shall be equivalent to the name of exactly one column of *ST*.

      NOTE 34 – A previous version of ISO/IEC 9075 allows <sort specification> to simply contain a <signed integer> to denote a column reference of a column of *T*. That facility no longer exists. See Appendix E, "Incompatibilities with ISO/IEC 9075:1992 and ISO/IEC 9075-4:1996", in ISO/IEC 9075.

Access Rules

None.

General Rules

1) All General Rules of Subclause 7.12, "<query expression>", in ISO/IEC 9075-2, apply to the <direct select statement: multiple rows>.

2) Let *Q* be the result of the <query expression>.

3) If *Q* is empty, then a completion condition is raised: *no data*.

4) If an <order by clause> is not specified, then the ordering of the rows of *Q* is implementation-dependent.

5) If an <order by clause> is specified, then the ordering of rows of the result is effectively determined by the <order by clause> as follows:

   a) Each <sort specification> specifies the sort direction for the corresponding sort key K_i. If ASC is specified or implied in the *i*-th <sort specification>, then the sort direction for K_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for K_i is descending and the applicable <comp op> is the <greater than operator>.

   b) Let X and Y be distinct rows in the result table, and let XV_i and YV_i be the values of K_i in these rows, respectively. The relative position of rows X and Y in the result is determined by comparing XV_i and YV_i according to the rules of Subclause 8.2, "<comparison predicate>", in ISO/IEC 9075-2, where the <comp op> is the applicable <comp op> for K_i, with the following special treatment of null values. Whether a sort key value that is null is considered greater or less than a non-null value is implementation-defined, but all sort key values that are null shall either be considered greater than all non-null values or be considered less than all non-null values. XV_i is said to *precede* YV_i if the value of the <comparison predicate> "XV_i <comp op> YV_i" is true for the applicable <comp op>.

   c) In the result table, the relative position of row X is before row Y if and only if XV_n precedes YV_n for some n greater than 0 (zero) and less than the number of <sort specification>s and $XV_i = YV_i$ for all $i < n$. The relative order of two rows for which $XV_i = YV_i$ for all i is implementation-dependent.

6) If Q is not empty, then Q is returned. The method of returning Q is implementation-defined.

Conformance Rules

None.